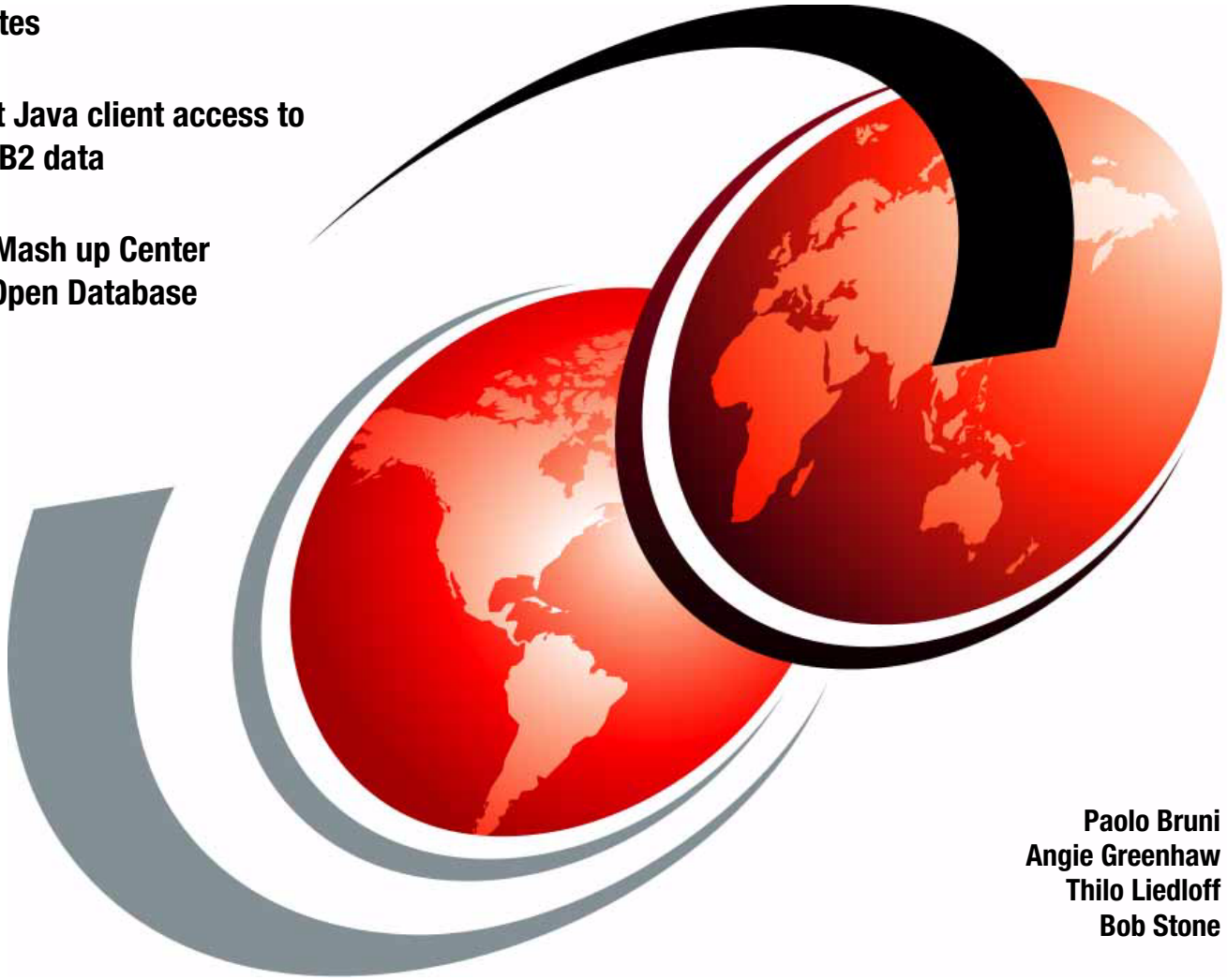
Information Management

IBM

IMS 11 Open Database

- Install IMS Open Database and its prerequisites
- Implement Java client access to IMS and DB2 data
- Integrate Mash up Center with IMS Open Database



Paolo Bruni
Angie Greenhaw
Thilo Liedloff
Bob Stone

Redbooks

ibm.com/redbooks

International Technical Support Organization

IMS 11 Open Database

August 2010

SG24-7856-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xvii.

First Edition (August 2010)

This edition applies to Version 11 of Information Management System (IMS) Transaction and Database Servers (program number 5635-A02) and IMS Enterprise Suite for z/OS, Version 1.1 (program numbers 5655-T60 and 5655-T61).

Contents

Figures

Tables

Examples

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
Cognos®
DataPower®
DB2 Universal Database™
DB2®
Distributed Relational Database Architecture™
DRDA®
Enterprise Workload Manager™
IBM®
IMS™
Informix®
InfoSphere™
iSeries®
MVS™
OMEGAMON®
Optim™
pureScale™
pureXML®
RACF®
Rational®
Redbooks®
Redpaper™
Redbooks (logo) ®
System z®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Cognos, and the Cognos logo are trademarks or registered trademarks of Cognos Incorporated, an IBM Company, in the United States and/or other countries.

Hibernate, Interchange, Red Hat, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

IBM® IMS™ Version 11 continues to provide the leadership in performance, reliability, and security that is expected from the product of choice for critical online operational applications. IMS 11 also offers new functions to help you keep pace with the evolving IT industry.

Using the new IMS Enterprise Suite application, developers with minimal knowledge of IMS Connect can start developing client applications to communicate with IMS.

With Open Database, IMS 11 also provides direct SQL access to IMS data from programs that run on any distributed platform, unlocking DL/I data to the world of SQL application programmers.

In this IBM Redbooks® publication, system programmers get steps for installing the new IMS components. The application programmer can follow scenarios about how client applications can take advantage of SQL to access IMS data.

We describe how to install prerequisites, such as IMS Connect and the Structured Call Interface component of Common Service Layer address space and document the set up of the three new IMS drivers:

- Universal DB resource adapter
- Universal JDBC driver
- Universal DL/I driver

Our scenarios use the JDBC driver for type-4 access from Windows® to a remote DL/I database and DB2® tables and extend it to use IBM Mashup Center. This process provides an effective web interface and integration with Open Database.

The team who wrote this book

This book was produced by a team of specialists from around the world working for the International Technical Support Organization at the Silicon Valley Lab, San Jose.

Paolo Bruni is an Information Management software Project Leader with the ITSO since 1998. He is based in the Silicon Valley Lab, San Jose. Paolo authors many IBM Redbooks publications about IMS, DB2 for z/OS®, and related tools and conducts workshops worldwide.

Angie Greenhaw joined IBM in 2000 after graduating from Arizona State University with a Bachelors degree in Computer Information Systems. She is currently an IT Specialist in the IBM IMS Advanced Technical Skills group, where she is a primary resource in the areas of IMS security, Dynamic Resource Definition, Common Service Layer, and Online Change. Prior to this role, Angie worked in IMS development, specializing in the Online Change function, contributing to new IMS functionality, and devising solutions as a Level 3 Service Representative in this same area. She also spent three years as the IMS Development Representative at SHARE. Angie wrote a white paper on the

topic of Global Online Change implementation and co-authored two other IBM Redbooks: *IBM IMS Version 10 Implementation Guide*, SG24-7526 and *IMS Version 11 Technical Overview*, SG24-7807.

Thilo Liedloff is a Field Technical Professional for IMS in IBM Germany since 2007. He has a Bachelors degree in Business Information Technology from the Baden-Wuerttemberg Cooperative State University Stuttgart, Germany. He has worked in the area of IMS and IMS Tools for three years. His specializations are in the areas of modern IMS Connectivity and IMS Application Development.

Bob Stone is a System z® IM Technical Pre-sales Professional for IBM in the United Kingdom (UK). He has eight years as a COBOL Application Developer, 18 years as an IMS DBA/Systems Programmer, and 10 years as a DB2 DBA working for a large number of commercial companies. He joined IBM in 2000 and spent eight years as a System z DB2 DBA and Systems Programmer supporting IBM internal systems before taking up his current role where he now specializes in IMS. He co-authored the book *DB2 for z/OS Tools for Database Administration and Change Management*, SG24-6420.

Acknowledgments

The authors thank **Rafael Avigad** for his contribution in written content. Rafael Avigad is a product architect and information developer with Fundi Software, in Perth, Western Australia. For the past five years, he worked on solutions for managing TCP/IP access to IMS and is helping develop current and future IMS tools graphical user interfaces. Before working at Fundi, Rafael developed operational support and billing systems for mobile telephony and ISPs.

Special thanks to Alison Coughtrie for making the basis for some example sources available and taking the time for a careful review. Special thanks to Kyle Charlet and Kevin Hite for the support throughout the project.

Thanks to the following people for their contributions to this project:

Rich Conway
Bob Haimowitz
Emma Jacobs
International Technical Support Organization

John Barmettler
Thomas Bridges
John Butterweck
Kyle Charlet
Quddus Chong
David Compton
Kevin Hite
Barbara Klein
Rose Levin
Nisanti Mohanraj
Danny Nguyen

Richard Tran
IBM Silicon Valley Lab

Kiran Challapalli
IBM India Software Lab

Gary Wicks
IBM Canada

Alison Coughtrie
IBM UK

Denis Gaebler
IBM Germany

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction to IMS

In this chapter, we provide an introduction to the contents of this book.

We describe how IMS continues to provide solutions to exploit the latest technologies to address customer requirements. We include background information about the SOA support and the Open Database architecture. The IMS Open Database provides distributed access to IMS database resources. It also helps to drive open standards and to open technology into IMS. The open standards that are introduced into this solution include the Java™ EE Connector Architecture, JDBC, and DRDA®. We also provide an overview of the IMS Enterprise Suite.

The topics that we discuss in this chapter are:

- IMS is open
- IMS and SOA
- IMS and Open Database
- The IMS Enterprise Suite

1.1 IMS is open

The IBM Information Management System (IMS) is the IBM premier transaction and hierarchical database management system and the product of choice for critical online operational applications and data where support for high availability, performance, capacity and integrity are key factors. IMS manages the world's mission-critical data and continues as a major player in the on-demand world. IMS customers are still growing in size and in number. As we move into the era of on-demand computing, IMS helps to lead the way by continuing to provide solutions to exploit the latest technologies to address customers' requirements for performance and availability, but also focusing on enterprise modernization through integration and open access with an on demand service-oriented architecture.

New application development tools and the IBM service-oriented architecture capabilities for IMS can help your business improve the speed and agility of its development efforts. Both IMS and the IMS SOA Integration Suite support your on demand systems and your distributed IMS application environment.

Despite many monumental improvements in recent years, IMS still faced significant restrictions that prevented it from fully supporting business growth and agility. Data is one of the largest assets in any business, and accelerating an organization's ability to share and deliver trusted information is essential to the success of that business. This is a competitive advantage. IMS data was not easily accessible and sharable. IMS Open Database is a new function in version 11 that addresses this need and takes on the challenge of both modernizing and standardizing IMS database access and application development.

To achieve this goal, IMS overcame two obstacles:

- Connectivity to the data
- Lack of industry standards as part of the data access model for IMS

As part of the Open Database solution IMS is rolling out a suite of universal drivers in support of the various standards and runtime containers. These drivers are referred to as:

- IMS Universal Database Resource Adapter for Java EE applications
- IMS Universal JDBC driver for Java SE applications
- IMS Universal DL/I driver for Java SE applications with more traditional DLI syntax

The term Universal is used because the drivers share the same common framework and can deploy in all of the supported platforms and runtimes. The drivers have environment detection protocols that are built into them so that application developers need not concern themselves with where the application is being deployed; instead, the drivers handle all runtime and platform-specific needs internally.

With Open Database, IMS 11 provides direct SQL access to IMS data from programs that are running on distributed platforms. Open Database reduces the complexity and processing that is associated with IMS data access unlocking DL/I data to the wide population of SQL application programmers.

The introduction of the new IMS Enterprise Suite Connect APIs for Java and C allows application developers, with minimal knowledge of IMS Connect, to start developing client applications to communicate with IMS. The Connect APIs is a simple, easy-to-use, lightweight programming solution for communicating with IMS transactions through IMS Connect.

1.2 IMS and SOA

Service-oriented architecture (SOA) is an architecture style that is centered around components or services that have standardized interfaces. It is a methodology of designing and running the software portion of an information technology infrastructure so that it supports the various individual and interrelated functions that are needed to operate a particular enterprise. SOA helps to bridge the business/IT gap and helps systems remain scalable and flexible as your business is growing and changing. SOA is focused on business processes, and although many legitimate approaches exist for software architecture, SOA is intended explicitly for business applications:

- Reusing
- Packaging
- Liberating business from the constraints of technology
- Services

For a more exhaustive discussion about IMS and SOA, refer to the IMS manuals and *Powering SOA Solutions with IMS*, SG24-7662.

1.2.1 The value of including existing IMS assets into SOA

There is great value in utilizing existing IMS assets in modernization projects:

- Existing IMS assets support core business processes and provide crucial information.
- Existing IMS assets contain billions of lines of valuable business rules.
- Using proven, time-tested IMS applications can significantly lower risk, cost, and time to market.
- The quality of the IMS system is recognized by all IMS users as fast, reliable, and mature.
- An IMS server is not a single point-of-failure. IMS has built-in recovery features, with take-over mechanisms, and it supports a Sysplex environment.

Implementing SOA is valuable because it adds a fast market driven responsiveness model to the traditional IMS application development and database creation.

When considering IMS assets, we must distinguish two areas or functions of IMS: the Transaction Manager (IMS TM) and the Database Manager (IMS DB):

- IMS TM or data communication controller (DCCTL) is the facility to link applications that are running as transactions to networks.
- IMS DB or Data Base Controller (DBCTL) interfaces with non-IMS communication controllers and traditionally supplies database services though the facilities of Database Resource Adapter (DRA) code.

Figure 1-1 on page 4 illustrates these two environments.

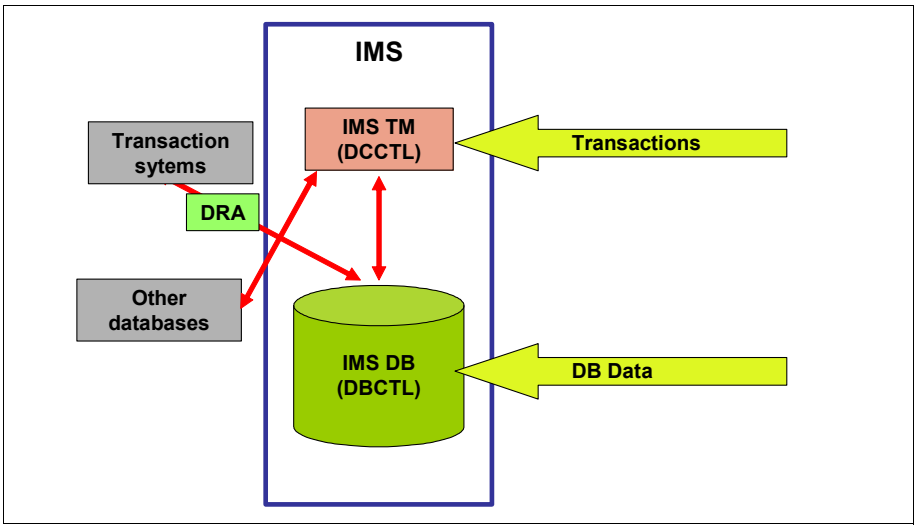


Figure 1-1 The IMS TM and IMS DB components of IMS

When considering IMS integration in SOA, we include both domains, TM and DB.

IMS provides the ability to use existing IMS transactions by making them available as callable Web Services. The solutions allow network access to and from the IMS host environment and also open IMS databases for access by non-IMS service requestors.

The components include:

- IMS Connect is the TCP/IP gateway to IMS applications, data, and operations and runs within a separate z/OS address space to the IMS control region. With IMS 11, IMS Connect is also an ODBM client that allows distributed applications to access any database in the entire IMSplex.
- The IMS TM Resource Adapter (TM RA) is a Java EE Connector Architecture (JCA) resource adapter that offers support to access existing IMS transactions from callable Web Services, Enterprise JavaBeans (EJBs), or even from HTML pages and servlets. IMS TM RA is an IBM WebSphere® Application Server-based solution. The tooling used in this solution is either IBM Rational® Application Developer (RAD) or WebSphere Integration Developer (WID), and they receive definitions of the input/output messages in either COBOL, C, or PL/I and generate all of the necessary artifacts and code.
- Another WebSphere Application Server-based solution is targeted at applications that are MFS based. MFS is much more complex than, for example a simple COBOL copybook, and as such IMS offers a specific solution for MFS. This solution offers tooling that consumes the MFS definition of the input/output messages and again generates the necessary artifacts. The MFS solution is an IMS TM Resource Adapter client so it leverages all of its functionality.
- Another solution offered is the IMS Enterprise Suite SOAP gateway, which offers direct SOAP access to existing IMS transactions. It is a non-WebSphere-based solution and does not require a Java EE (Java Platform, Enterprise Edition) container. There is tooling that is specific to this solution in Rational Developer for System z that consumes input/output message definitions and again generates all of the necessary artifacts for exposing an IMS transaction as a callable SOAP service.
- Although the TM-based solutions put focus on leveraging existing IMS transactions, the DB-based solutions are focused on new IMS application development. These solutions offer direct IMS database access from a variety of environments. IMS gives WebSphere,

CICS®, DB2, and even application developers the ability to access IMS database assets using industry-standard programming models and APIs.

- Two new IMS JMP and JBP-dependent regions were introduced. A Java Message Processing (JMP) region is analogous to an MPP region, and a Java Batch Processing (JBP) region is analogous to a non-message driven BMP region. The fundamental difference is that with the new regions, IMS now can fully house and maintain a Java Virtual Machine (JVM), which enables IMS to now effectively process Java workloads. In addition to this, IMS offers the Java class libraries, which contain a complete API for Java developers to use. The Java libraries offer a JDBC driver for IMS, which can process both SQL and XQuery expressions.

  These same Java libraries can be utilized from several runtime environments:

  - WebSphere Application Server
  - DB2 stored procedures on z/OS
  - CICS using their JCICS API

- The IMS Open Database Access (ODBA) and DRA modules offer Java libraries the ability to access IMS databases from a non-IMS environment.
- For the WebSphere Application Server environment, IMS offers another JCA resource adapter, the IMS DB Resource Adapter. The difference with this adapter, as opposed to the IMS TM Resource Adapter, is that all of the business logic is in the EJBs themselves. With the IMS DB Resource Adapter, there is no IMS-dependent region involved at all.
- With IMS 11, the IMS Universal drivers, part of the Open Database solution, are software components that provide Java applications with access to IMS databases from z/OS and from distributed environments through TCP/IP.
- For all of these solutions, IMS also offers tooling support through the DLIModel utility, which is an Eclipse-based GUI tool that is now part of the Enterprise Suite offering. It offers visualization of IMS databases. It consumes PSB, DBD, and even COBOL copybook source to visualize all of the PCBs in a particular PSB. Information, such as hierarchies, segments, fields, and field types, are captured. With respect to new application development, the utility also generates database metadata definitions that are consumed at runtime by the Java libraries, which enables the libraries to convert an SQL query into a native IMS DL/I call.

1.2.2 IMS Connect and IMS Connect Extensions

Starting with IMS Version 9, IMS Connect is delivered as an integral component of IMS. It performs the vital function of connecting IMS to the TCP/IP world.

IMS Connect Extensions is a key tool for managing access to IMS through IMS Connect:

- Key benefits:
  - Provides event collection and instrumentation for IMS Connect
  - Streamlines operational management of IMS Connect and its clients
  - Assists in the development of TCP/IP clients and the transition to an SOA
- Principal users:
  - IMS tuning specialists, application developers, operators, and administrators

One example of using IMS Connect Extensions is in its assistance to monitor your IMS TCP/IP network flow. OMEGAMON® for IMS on z/OS, as a Real-time monitoring tool for IMS Connect, uses the Connect Extensions Publisher API where it obtains IMS Connect event records through the API.

1.2.3 The IMS SOA Integration Suite

The IMS SOA Integration Suite is a collection of IMS middleware functions and tools that support your IMS on demand systems and your distributed IMS application environment. It leverages the utilization of your existing assets and running systems to integrate IMS capabilities in a SOA environment by providing these capabilities:

- Provides access to IMS transactions and data from any web connection.
- Modernizes your IMS applications and enables them to operate with other clients, such as Microsoft®.NET or SAP clients in a service-oriented architecture.
- Integrates business logic that is embedded in your existing IMS applications with other IT systems, both within your enterprise and in the supply chain.
- Improves development time by using Java instead of PL/I, COBOL, or Assembler.
- Accesses your IMS data directly for use by your applications from environments, such as DB2, CICS, and WebSphere Application Server.
- Stores and retrieves your XML content directly in IMS without any intermediate steps and exchanges data with other systems by using established schemas.

The following tools and functions support access to IMS transactions:

- IMS Enterprise Suite Connect APIs
- IMS Enterprise Suite SOAP Gateway
- IMS TM resource adapter
- IMS MFS Web Solutions
- IMS Web 2.0 solutions for IBM Mashup Center

The following tools and functions support access to IMS data:

- IMS Enterprise Suite DLIModel utility plug-in
- IMS solutions for Java development
- IMS XML DB
- IBM Web 2.0 solution for IBM Mashup Center

In 1.4, “The IMS Enterprise Suite” on page 12, we provide an overview of the functions included in Enterprise Suite. For more information about all of the components of the IMS SOA Integration Suite, see:

http://www.ibm.com/software/data/ims/soa-integration-suite/

IMS TM Resource Adapter

The IMS TM Resource Adapter (TM RA), previously known as IMS Connector for Java, is part of the IMS SOA Integration Suite of middleware functions and tools. Using the IMS TM Resource Adapter you can quickly and easily create Java applications that access new and existing IMS transactions over the Internet. Using TM RA within a WebSphere or Rational-family development environment, you can:

- Develop components of business processes in support of SOA.
- Create Java EE applications from Java beans.

The development version of the IMS TM Resource Adapter is included in the following integrated development environments:

- Rational Application Developer for WebSphere Software
- WebSphere Integration Developer
- WebSphere Transformation Extender
- Rational Developer for System z (formerly known as WebSphere Developer for System z)
- Rational Software Architect

You can download the runtime component of the IMS TM Resource Adapter from the web site:

http://www.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#downloads

The value of using IMS TM Resource Adapter

The IMS TM Resource Adapter implements the Java EE Connector Architecture (J2C), which connects Enterprise Information Systems (EIS), such as IMS to the Java EE platform. The Java EE Connector Architecture provides your applications with the qualities of service that a Java EE application server can provide, such as connection, transaction, and security management, which allows for:

- You can use the IMS TM Resource Adapter with a Java EE server, such as IBM WebSphere Application Server, when a Java application accesses an IMS transaction that is running on a host IMS system. The IMS TM Resource Adapter also enables an IMS application to act as a client to invoke applications in a Java EE server.
- Although the IMS TM Resource Adapter is intended for use primarily by Java applications or Web Services that submit transactions to IMS, the IMS TM Resource Adapter can also be used by services that submit IMS commands to IMS.

IMS TM RA provides tooling support for development of Java EE applications, Web Services, and business processes that access IMS transactions in various Rational and WebSphere-integrated development environments.

IMS TM RA also provides programming for deployment to the WebSphere Application Server and WebSphere Process Server (WPS) runtime environment on many platforms, including z/OS and Linux® on System z.

1.2.4 DataPower and IMS

DataPower® is a powerful solution for XML acceleration, XML security, monitoring, and for managing SOA environments.

Included in the DataPower firmware V3.6.1 is a feature called *IMS Protocol Support* that adds support to allow Multi-Protocol Gateway services to accept IMS connections from clients and connect to IMS-based applications. This functionality provides:

- An IMS Connect proxy to IMS Connect clients. The use case is for existing IMS Connect clients who want to make in-flight modifications to headers and payloads without changing the client or IMS.
- Web Service facade to IMS Connect transactions. The use case is to make use of the strong Web Service features in DataPower to quickly enable Web Service support for IMS Connect.

Figure 1-2 on page 8 presents the DataPower hardware components and their primary roles.

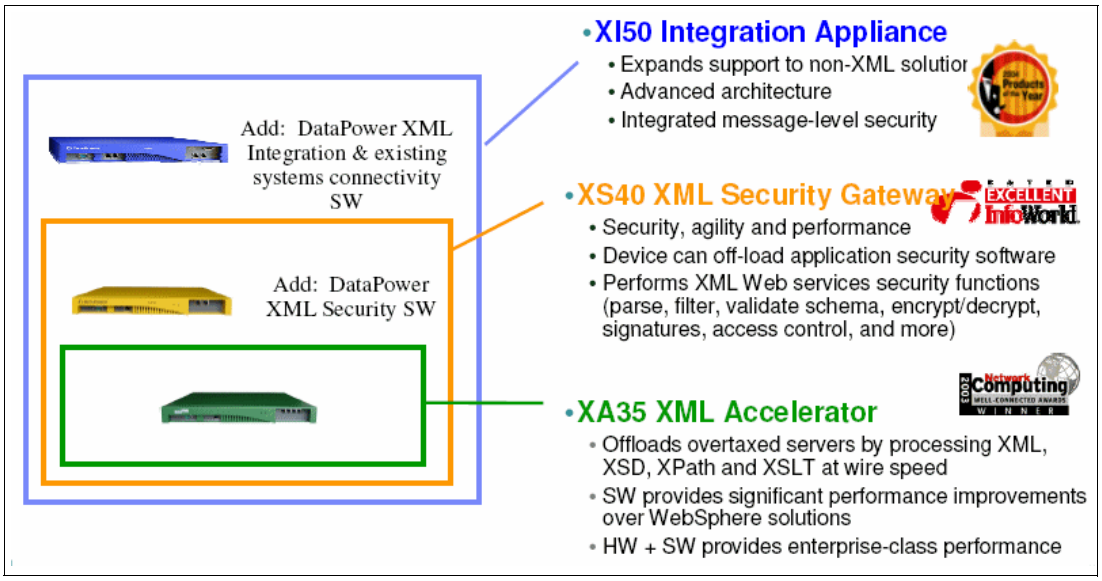


Figure 1-2 DataPower components

1.3 IMS and Open Database

IMS 11 ensures that you can continue to rely on IMS and its hallmarks of reliability, availability, matchless scalability, and performance. You can take advantage of:

- Dynamic commands to query and change IMS Transaction Manager and IMS Database Manager resources, to create points of consistency, and to reduce the time that databases are offline.
- User exit services enhancements to define multiple instances and to dynamically refresh modules.
- Enhanced syntax checking.
- System performance enhancements and improved availability through IMS Fast Path Buffer Manager, Application Control Block library, and Local System Queue Area storage reduction, which utilize 64-bit storage and frees up valuable ECSA.

When running core applications that are at the heart of business processing, most large corporations worldwide continue to depend on IMS. Today's enterprise IT needs are more closely tied to the business than ever before. Businesses require efficiency to meet the cost challenges and responsiveness the global economy demands. IMS 11 provides an easier than ever roadmap that supports your business growth for years to come.

The Open Database capability reduces the complexity and processing that is associated with IMS data access. IMS Connect now provides simplified TCP/IP access to *both* IMS transactions and data.

This support for easier integration and open access can also help with industry regulations compliance and internal controls, and can enable you to more rapidly develop and deploy new applications and services.

The IMS Open Database provides distributed access to IMS database resources. It also helps to drive open standards and open technology into IMS. The open standards that are introduced into this solution include the Java EE Connector Architecture, JDBC, and DRDA.

Historically, the IMS database was a closed architecture, and by opening it up, IMS is positioned for the future as it pertains to industry standard access APIs and the emerging SOA market.

The business challenges that Open Database addresses are:

- Data can be difficult to access outside of the IMS environment. Clients might want to participate in the emerging SOA market.
- Traditional IMS support skills are becoming increasingly rare and difficult to acquire and train.
- Takes too long to deploy new applications and to enhance existing applications.

The value of using IMS Open Database

The ability to access IMS data from outside of the IMS environment simplifies the process of developing new applications that use existing investment in IMS data. Also, it reduces costs by providing distributed access to IMS database resources through industry-standard interfaces.

The distributed access function offers two distinct types of IMS database resource distribution:

- The distribution of IMS database resources across LPARs in an IMSplex. IMS data can be accessed through TCP/IP from a Java EE application server that resides on a z/OS platform that is on a separate logical partition (LPAR) from the IMS subsystem.
- Pure distribution, which means that IMS database resources are now directly accessible from non-mainframe platforms, which includes full distributed transaction processing and two-phase commit semantics.

All of this is accomplished by three main components:

- Client-side libraries that implement the industry-standards interfaces and protocols.
- IMS Connect, which processes the distributed requests.
- The use of the Open Database Manager (ODBM) address space.

Figure 1-3 on page 10 introduces the Open Database environment that is available with IMS Version 11.

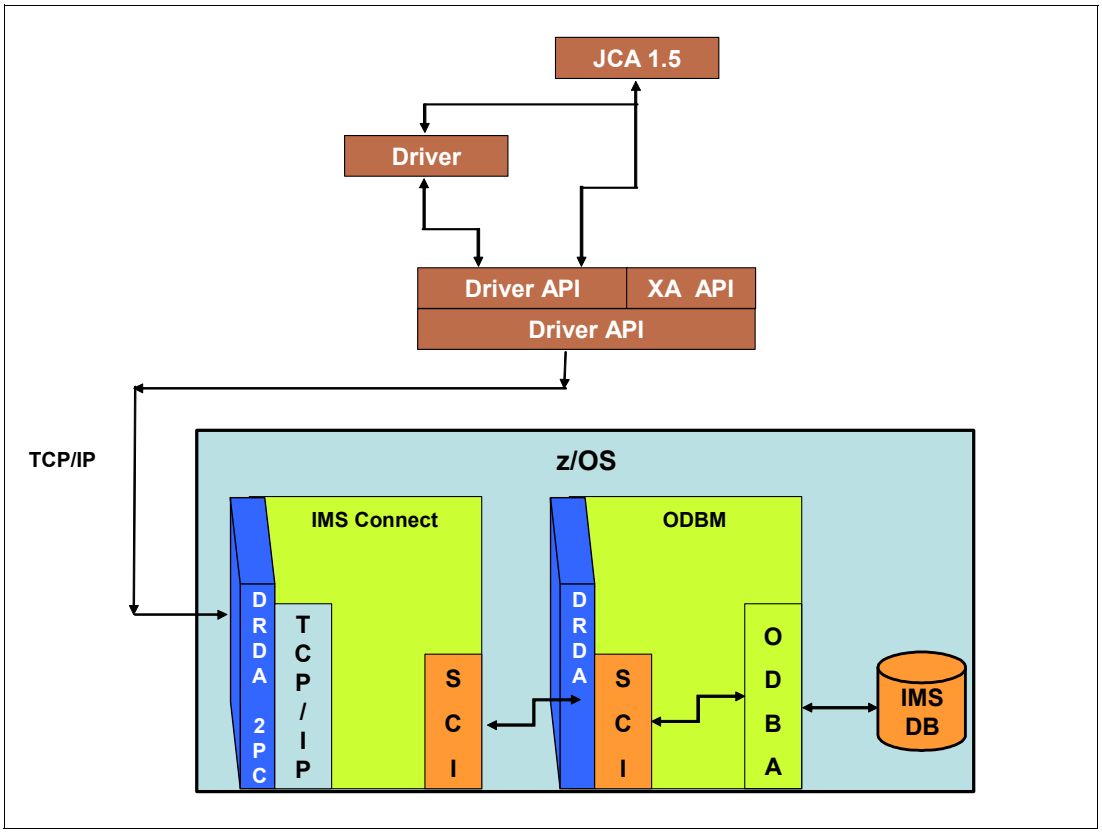


Figure 1-3 Open DB environment

For details about Open Database environment, see Chapter 2, “Open Database architecture” on page 19.

Data is one of the largest and most valuable assets of any business. An organization’s ability to share and deliver trusted information is not only essential to the success of that business, but provides a competitive advantage. However, IMS data has not always been easily accessible and shareable in the way that supports the growth and agility of businesses. To meet these needs, IMS had to overcome two obstacles: connectivity to the data and the lack of industry standards as part of the data access model for IMS. The IMS Version 11 Open Database solution addresses the challenges of modernizing and standardizing both IMS database access and application development by providing an integrated distributed data access solution. Now you can more easily integrate IMS assets with other products and platforms across your enterprise and the Internet.

The IMS Universal drivers

The IMS Universal drivers, part of the Open Database Solution, are software components that provide Java applications with connectivity and access to IMS databases from z/OS and from distributed environments through TCP/IP.

IMS provides three Universal drivers that support multiple standards and runtime environments:

- IMS Universal DB Resource Adapter: A JCA-compliant resource adapter that provides all of the services that the JEE platform provides, including connection, transaction, and security management

- IMS Universal JDBC driver: A Java Database Connectivity (JDBC) driver that supports access to IMS data by using SQL calls
- IMS Universal DL/I driver: An IMS-specific Java API for DL/I that can access IMS data using Java methods that are based on IMS DL/I semantics

The term Universal is used because the drivers share the same common framework and can be deployed in multiple platforms and runtime environments. Because the drivers have environment-detection protocols built into them, application developers do not need to be concerned with where the application is deployed–the drivers handle all runtime environment and platform-specific needs.

The IMS Open Database solution focuses on Java programming skills. The ability to tap into this skill base opens significant opportunities for new IMS application development. The standard data access interface in Java today is JDBC. JDBC offers support for SQL, which is used almost exclusively across all major database management systems. The IMS Open Database solution provides an implementation of JDBC coupled with an engine for SQL processing. If you want to develop an IMS application in Java, you now have several options, as shown in Figure 1-4.

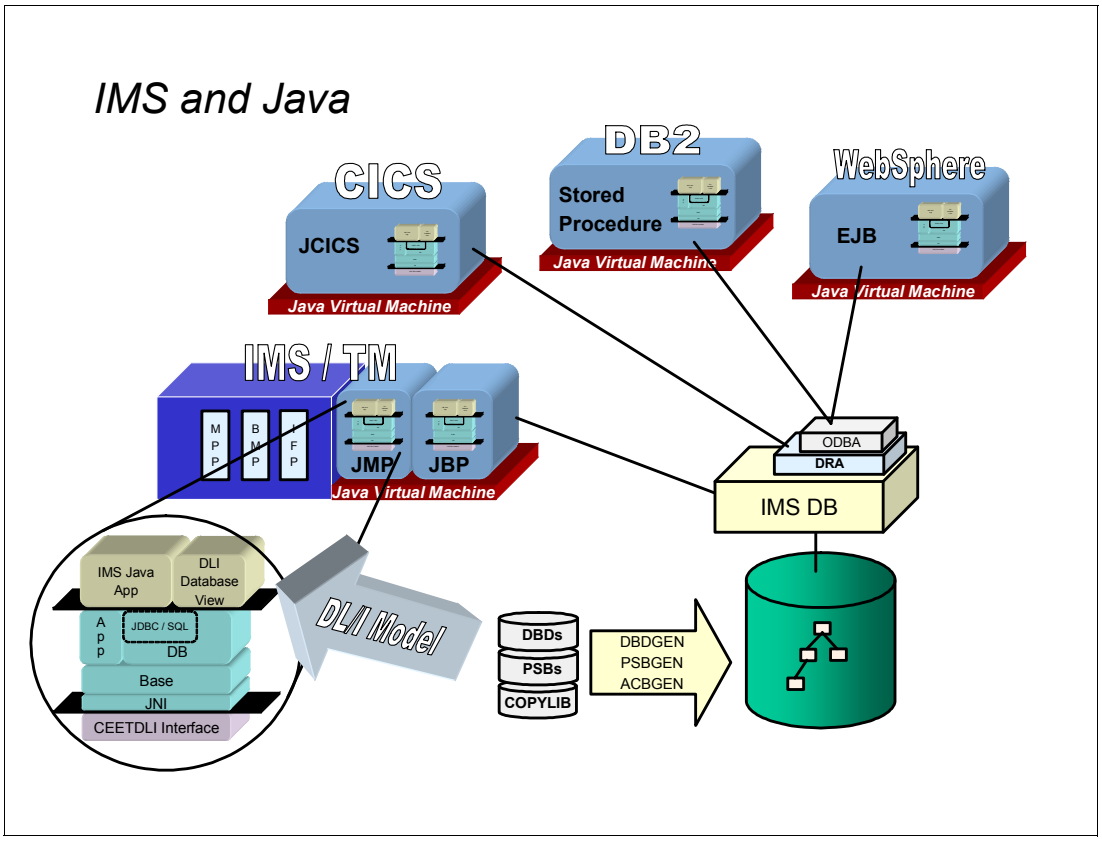


Figure 1-4 IMS and Java: The options

The main choice for many application architects is enterprise application development in tightly managed application servers. The Java EE and Java Connection Architecture standards are built around integration into these server runtime environments. WebSphere Application Server, which is the IBM Java EE server, adheres to these standards, as does the IMS Open Database solution. Application developers can now deploy their applications into the WebSphere Application Server runtime environment and take full advantage of its management, connection pooling, and security capabilities.

IMS Connect has been the gateway for TCP/IP access to IMS transactional resources. The IMS Open Database solution enhances IMS Connect so that it now provides access to IMS database resources using a new CSL address space: Open Database Manager (ODBM).

To provide programmatic access to IMS data, the Universal drivers can be deployed in the following runtime environments:

- IMS Java dependent regions
- WebSphere Application Server for distributed platforms
- WebSphere Application Server for z/OS
- CICS
- DB2 and DB2 for z/OS stored procedures
- Standalone Java SE

The drivers can run on both distributed and z/OS platforms. TCP/IP is used to access IMS data from distributed platforms, which includes Windows, Linux, AIX®, Sun Solaris, UNIX®, and Linux for System z. Pre-existing assembler interfaces are used to access IMS data from z/OS runtime environments.

The Open Database solution allows authentication of all clients at various layers, for example, in a WebSphere Application Server (Java EE) runtime environment, you can use encrypted credentials to manage the security credentials instead of having the application manage them. RACF® on z/OS (or an equivalent security product) is used to both authenticate the user and to ensure that the user has the authorization to the PSB. The Secure Sockets Layer (SSL) protocol is supported for the type-4 Universal drivers to ensure that your communication layer is encrypted. Rest assured that the Open Database solution offers a robust security model to fit your needs.

1.4 The IMS Enterprise Suite

Developing an IMS Connect client application to access IMS transactions can be challenging for application developers. To implement complex business scenarios, an IMS Connect application developer needs specific programming skills to code an IMS Connect client, including in-depth knowledge of IMS Connect protocols, TCP/IP socket programming, message and header formats, and how to set header fields.

With the introduction of the new IMS Enterprise Suite Connect APIs for Java and C, application developers only need minimal knowledge of IMS Connect to start developing client applications to communicate with IMS. The Connect API is a simple, easy-to-use, lightweight programming solution for communicating with IMS transactions through IMS Connect. Developers can use a high-level programming language to write IMS Connect client applications. In addition, developers have the flexibility to run the applications standalone, without the overhead of an enterprise application server environment.

The components of the IBM IMS Enterprise Suite V1.1 fully integrate with IBM WebSphere, IBM Rational, and industry tooling. They utilize a common programming model for a service-oriented architecture based on standards, such as XML, SOAP, Java, JDBC, and other emerging standards. The IMS Enterprise Suite V1.1 release includes:

- Connect APIs
- Java Message Server API
- SOAP Gateway
- DLIModel Utility plug-in

1.4.1 IMS Enterprise Suite Connect APIs

IMS Enterprise Suite Connect APIs for Java and C enable applications to more easily develop client applications that communicate with IMS. Distributed platforms can now more easily connect to IMS. The design, development, and testing of IMS access for client TCP/IP applications has never been simpler. The Connect APIs provide a simple, standard way to describe TCP/IP socket connections, interaction protocols, message headers, and data. Developers can use preconfigured values to create connections and interactions rather than manually setting these values.

The Connect APIs is a lightweight programming solution in Java that requires minimal set up, is easy to configure, and requires no additional tooling. Developers can use Java support for code portability, exception handling, and logging. To accelerate development, developers can take advantage of an integrated development environment (IDE) for Java development, such as IBM Rational Application Developer or IBM Rational Developer for System z.

Figure 1-5 shows how the Connect APIs hide the complexity of communication protocols for interactions with IMS Connect behind a simple programming interface.

The client application sets connection and interaction properties by loading from a profile or configures the property values programmatically. To perform an interaction, the client application simply issues an execute function call. The interaction types that are supported by the Connect APIs are SENDRECV, RESUMETPIPE, and several SENDONLY interaction types.

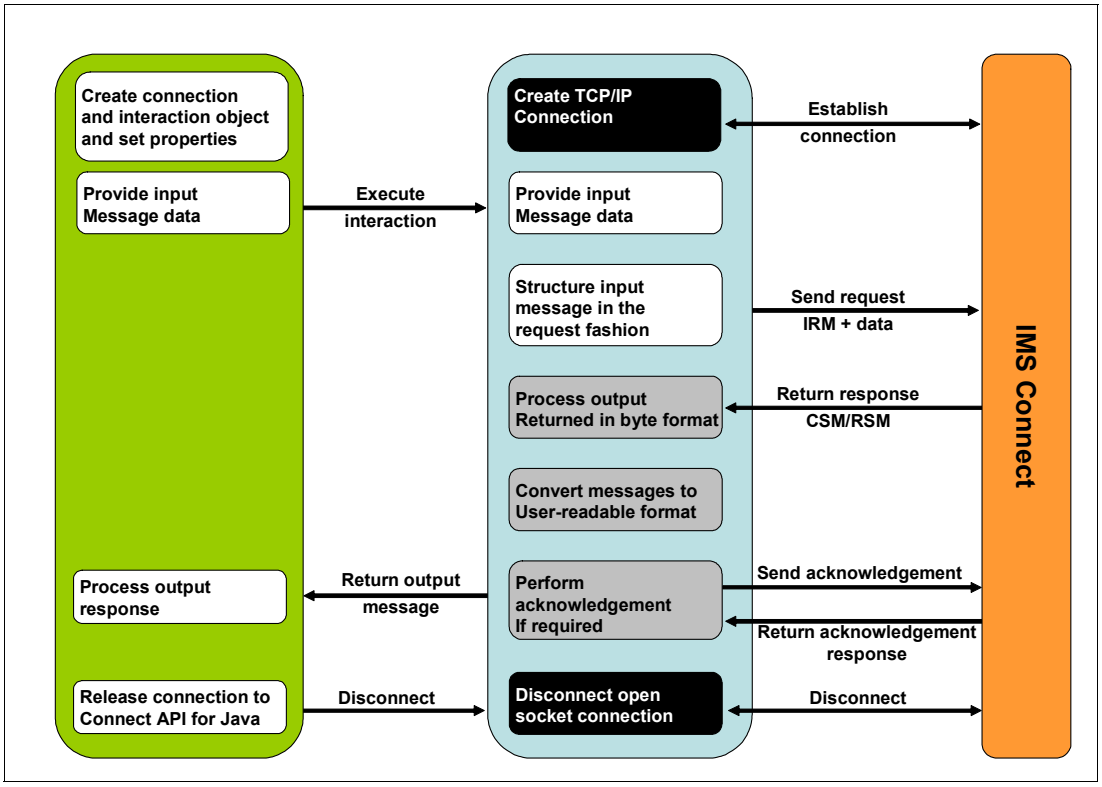


Figure 1-5 The IMS Enterprise Suite Connect APIs simplifies client interactions with IMS

The Connect APIs handle the work of establishing a connection, creating an IMS Connect input message, sending the input message to IMS Connect, retrieving any output message that IMS Connect returns, and storing the message in a data structure for easy client-application retrieval.

The Connect APIs also support profiles. Profiles are plain text files that contain property name-value pairs that can be predefined for specific connection and interaction scenarios and reused to load connection and interaction property values in separate application clients. Developers can also configure the connection and interaction properties dynamically in the client by using property-setter methods.

1.4.2 Java Message Server API

Using the industry-standard Java Message Server (JMS) API application developers can define a common set of messaging concepts and programming strategies that are supported by all JMS technology-compliant messaging systems, which increases their overall productivity and addresses skills issues. The JMS API can be used for synchronous callout from an IMS Java application.

1.4.3 IMS Enterprise Suite SOAP Gateway

IMS Enterprise Suite SOAP Gateway (formerly called IMS SOAP Gateway) is an XML-based connectivity solution that enables existing or new IMS applications to communicate outside of the IMS environment using SOAP message protocol to provide and request services independently of platform, environment, application language, or programming model.

Figure 1-6 on page 15 illustrates the IMS SOAP Gateway deployment and runtime environment. IMS SOAP Gateway uses RDz to generate both the correlator and WSDL files that are used when deploying Web Services within IMS SOAP Gateway. An IMS SOAP Gateway Deployment utility is included for you to set up properties to deploy and maintain IMS Web Services. Also, an IMS SOAP Gateway Administrative Console is available to list the deployed Web Services when the server is started.

IMS SOAP Gateway interfaces with IMS Connect using TCP/IP protocols but uses SOAP or HTTP or HTTPs when communicating with SOAP clients.

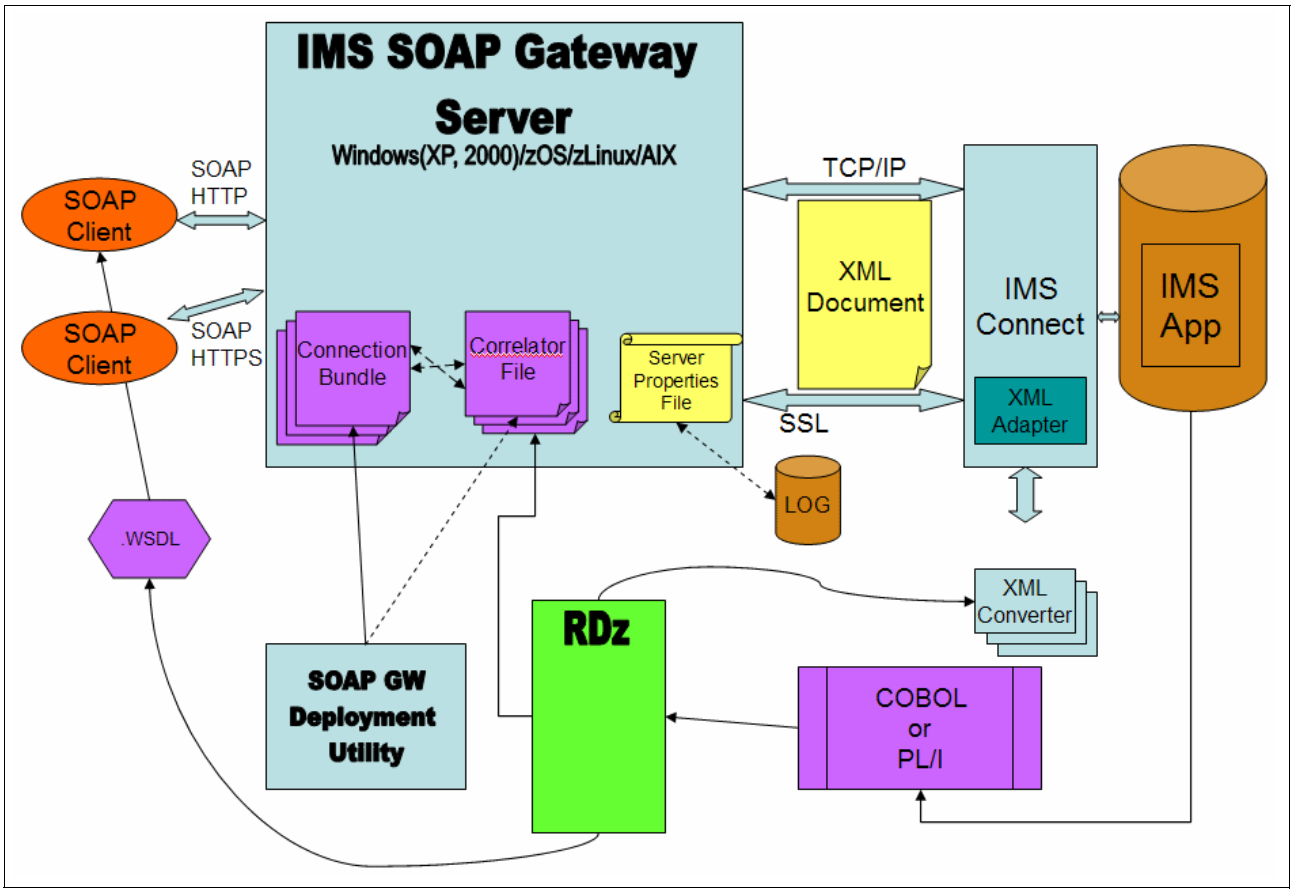


Figure 1-6 IMS Enterprise Suite SOAP Gateway development and runtime environment

The value of using SOAP Gateway

IMS Enterprise Suite SOAP Gateway is a light-weight Web Services solution that enables IMS applications to interoperate in a SOA environment without needing a full-blown application server (for example, Java EE Server). One typical usage scenario of providing Web Services with the IMS SOAP Gateway is to enable Microsoft.NET client applications or intermediary servers that submit SOAP requests into IMS to drive business logic transactions.

Through SOAP, IMS SOAP Gateway provides and requests services that are independent of platform, environment, application language, or programming model:

- It enables IMS application assets as Web Services.
- It allows non-WebSphere customers to reuse existing and to create new IMS-based business logic.
- It operates with any types of client application using SOAP/HTTP protocols.

Generated IMS service definitions (that is Web Services Description Language files) can be published or exposed to an UDDI directory for businesses to publish their offerings and for users to discover their needs. You can retrieve IMS WSDL files out of the UDDI directory and fit them into a tool (such as Microsoft.Net or Apache Axis server tools) to generate SOAP messages to be sent to the host to run existing IMS applications.

1.4.4 IMS Enterprise Suite DLIModel utility plug-in

The IMS Enterprise Suite DLIModel utility plug-in (formerly called IMS DLIModel utility) transforms your IMS database information (program specification blocks, database descriptions, and COBOL copybooks or PL/I includes) into application-independent metadata that can be used for Java application development. Without the DLIModel utility, the IMS-Java user must manually create Java classes that describe the metadata (for example, structure, segment layouts, and so on) of the IMS databases that are to be processed. The DLIModel utility creates these classes from PSB and DBD sources plus optionally, high-level language sources that more fully describe segment field layouts. The DLIModel utility runs as a plug-in to Eclipse, WebSphere Developer for System z, RAD for WebSphere, and RDz.

The value of using the DLIModel utility

The business goal of the DLIModel utility is to ease IMS application development. Using the DLIModel utility you can transform your IMS database information (program specification blocks, database descriptions, and COBOL copybooks) into application-independent metadata. In addition to creating metadata, using the IMS DLIModel utility you can:

- Generate XML schemas of IMS databases, which are used to retrieve XML data from or store XML data in IMS databases.
- Incorporate additional field information from COBOL copybooks.
- Incorporate additional PCB, segment, and field information.
- Override existing information.
- Generate a DLIModel report, which is designed to assist Java application programmers in developing applications based on existing IMS database structures.
- Generate an optional DLIModel trace log.

Figure 1-7 on page 17 illustrates the input and output flows from the DLIModel utility.

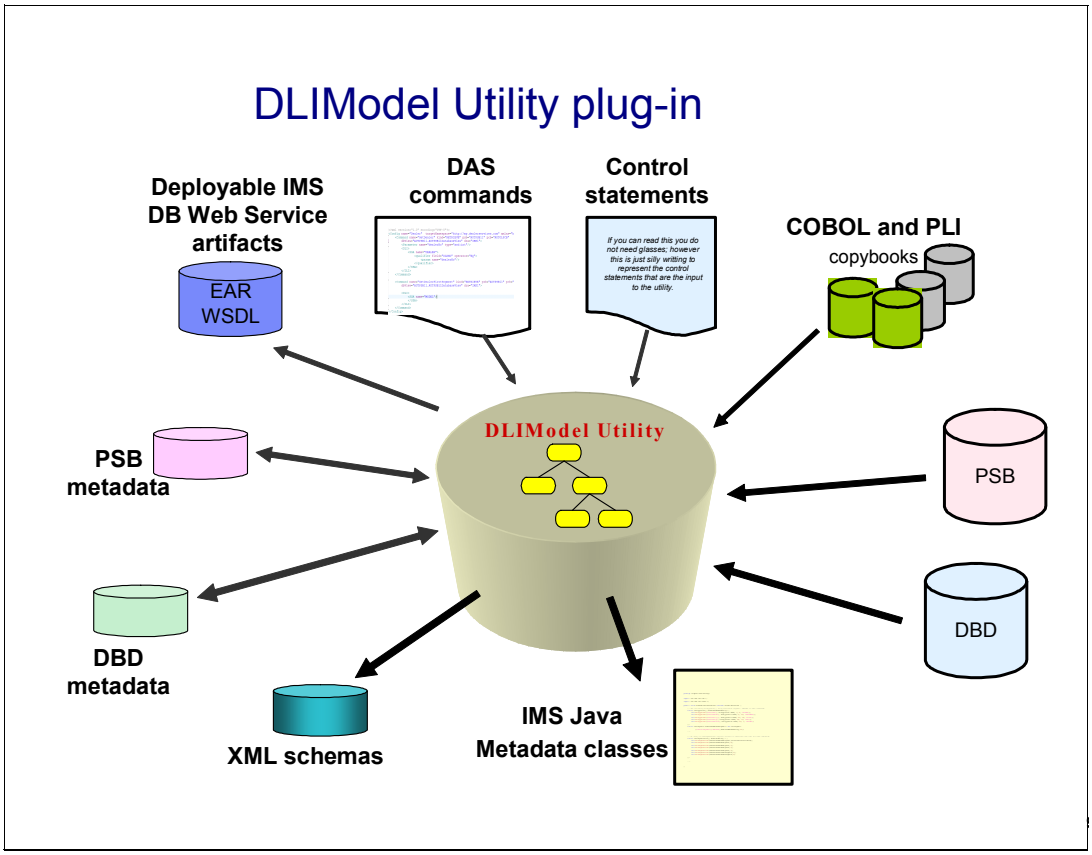


Figure 1-7 DLIModel Utility input and output flow

2

Open Database architecture

In this chapter we describe the architecture of an IMS Open Database environment.

The topics that we discuss in this chapter are:

- Why IMS Open Database
- Accessing IMS DB in Versions 9 and 10
- Evolution in IMS 11
- IMS 11 architecture
- Open Database functions
- IMS 11 Universal drivers

2.1 Why IMS Open Database

IMS Open Database is a new function in IMS 11 that takes on the challenge of modernizing IMS Database access and application development. It provides an environment that manages access to online IMS databases from anywhere in the enterprise. It supports open-standards for connectivity to online IMS databases:

- Within a z/OS LPAR
- Across z/OS LPARs
- From distributed platforms

IMS Open Database addresses two significant bottlenecks for business growth:

- Connectivity: IMS DB has been historically grounded to the mainframe with ways to get to it but none straightforward and simple.
- Application development: Even when connectivity is not an issue, the skills are not readily available to develop new application workload. DL/I is not industry standard and skills are not plentiful.

IMS 11 rolls out a complete suite of Universal drivers in support of IMS database connectivity and programmatic access. The intent is to access IMS in a uniform way using the most relevant industry standards from any platform and from within the most strategic runtimes. A standards-based approach opens a lot of growth and expansion opportunity. The fundamental communication protocol for communicating with IMS Connect is the industry standard Distributed Relational Database Architecture™ (DRDA) protocol. Each Universal driver supports both type-4 and type-2 connectivity in all supported runtimes; therefore, there is no need to learn another driver's semantics to toggle between environments and desired connectivity because it is all built into the framework. Distribution of resources within an IMSplex is included. The idea is to extend the reach of IMS by extending the data. IMS DB metadata is exposed by use of the standard JDBC API and therefore can be consumed and visualized by JDBC tooling. By allowing inspection of metadata, the next step is query. Query syntax uses standard query language syntax.

IMS Open Database offers direct distributed access to IMS database resources. The distributed nature is two-fold:

- At the IMSplex level, it allows cross-LPAR access to any IMS database in the IMSplex.
- At the pure distributed level, it allows non-mainframe (for instance, Windows OS) access directly to IMS database resources through industry standard interfaces.

This enhancement extends IMS Connect as the gateway to IMS DB. It adds a new Common Service Layer address space that manages connections to the IMS ODBA interface. This enhancement improves application access to IMS.

IMS has seen an increased number of requests for distributed access to all database types. IMS Connect is currently the gateway to IMS TM.

Distribution of database assets comes in two flavors:

- Distribution within an IMSplex: Applications on one LPAR can access an IMS database on another LPAR.
- Distributed access from non-System z platforms: Applications on a non-System z platform can have direct IMS DB access without needing an IMS transaction to proxy the data.

The Universal drivers (JCA, JDBC, DL/I) allow both distributed as well as local (CICS, IMS, WebSphere Application Server on System z, DB2 for z/OS) access to IMS databases.

Figure 2-1 shows an IMS 11 Open Database overview.

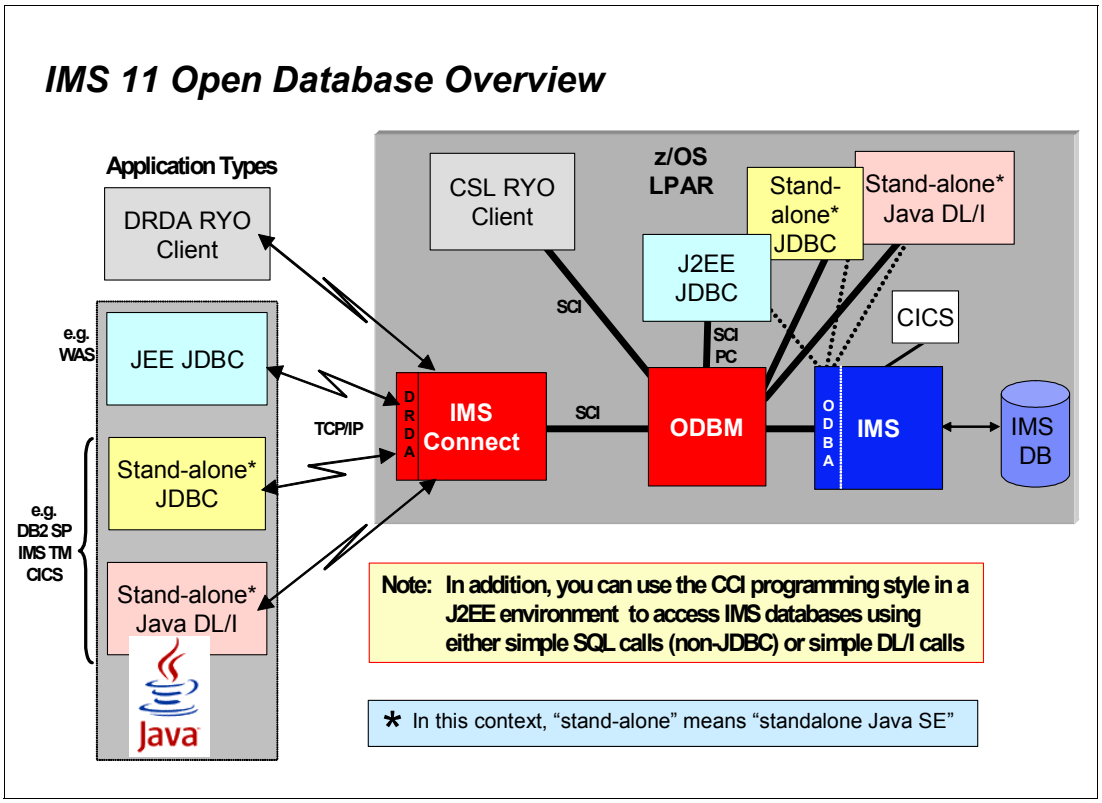


Figure 2-1 IMS 11 Open Database overview

2.2 Accessing IMS DB in Versions 9 and 10

The interfaces to Online IMS DB in IMS 9 and 10 are:

- Open Data Base Access (ODBA), which is used by WebSphere Application Server on System z and DB2 stored procedures, and with these Resource Recovery Services (RRS) is required as the synchronization point coordinator.
- The transaction manager Customer Information Control System (CICS) uses CCTL as its interface with CICS itself being the synchronization point coordinator, and there is no requirement for RRS.
- Both ODBA and CCTL use the Database Resource Adapter (DRA). Some IMS modules reside in the ODBA application or in CICS address spaces, in particular the DRA Startup Table (assembled DFSPRP macro).
- IMS Java provides two IMS DB *Classic* drivers. The drivers are now referred to as Classic to differentiate them from the *Universal* ones that are delivered with IMS 11. The Classic drivers are still delivered and supported with IMS 11; however, the Universal drivers are recommended:
  - The IMS classic JDBC driver provides support for writing JDBC applications for CICS, DB2 stored procedures, and WebSphere Application Server for z/OS environments. This IMS classic JDBC driver supports a selected subset of the full facilities of the JDBC 2.1 API. The IMS 11 Universal JDBC driver provides improved support for standard SQL syntax.
  - The IMS Java JDBC Resource Adapter for JEE environments provides support.

The IMS Universal DB resource adapter is built on industry standards and open specifications and provides more flexibility and improved support for connectivity, data access methods, and transaction processing options. Use the IMS Universal DB resource adapter to develop Java EE applications that access IMS from WebSphere Application Server.

Figure 2-2 shows methods of accessing online IMS DB in V9 and V10.

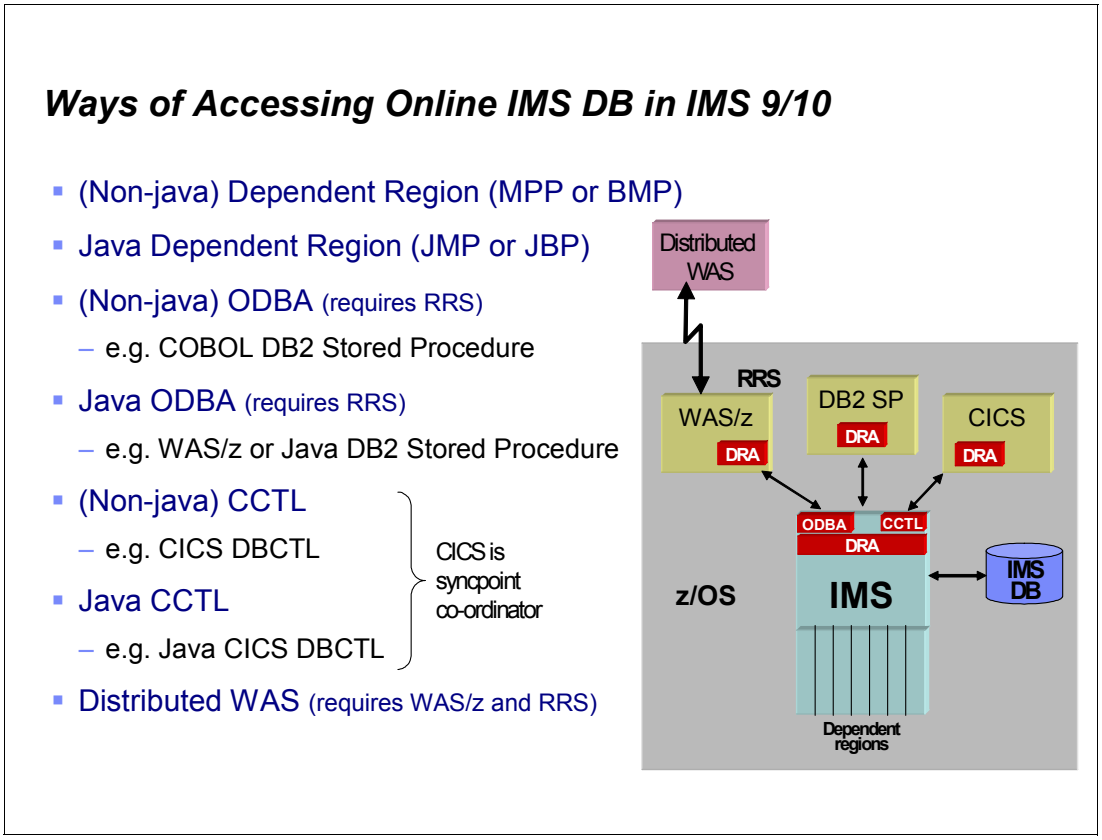


Figure 2-2 Methods of accessing online IMS DB in V9 and V10

There are several problems that are associated with the IMS 9/10 methods:

- If an ODBA application terminates in the middle of a DL/I call, there is the possibility of crashing IMS with a 113 abend.
- When WebSphere Application Server for z/OS is used, it must run in the same LPAR as IMS, which can affect machine capacity and license charges.
- If remote WebSphere solutions are being used, WebSphere Application Server for z/OS is needed, and a *helper* EJB running in WebSphere Application Server for z/OS relays the remote request to ODBA.
- Distributed applications accessing IMS DB must execute on WebSphere Application Server. There is no support for other distributed platforms.

2.3 Evolution in IMS 11

Our intent in this section is to show the topology prior to IMS 11 and illustrate the evolution to IMS 11, pointing out the enhancements at each step. As a point of fact, WebSphere

Application Server for z/OS cannot take advantage of the cross-LPAR feature of ODBM unless WebSphere Application Server itself embraces Structured Call Interface (SCI). Applications can use the out-of-the-box compatibility mode to use AERTDLI and have those calls routed to an ODBM, which still prevents the U113 abend; however, WebSphere Application Server and the ODBM address space must still be on the same LPAR (It is just an illustrative example showing what can be possible with WebSphere Application Server for z/OS as an ODBM client).

The IMS 9 and 10 solution (whether or not we are talking about distributed or local access to IMS DB) leverages Open Database Access as the API to access IMS database resources. ODBA can make address space to address space calls (PC calls) in the same logical partition, which means that the ODBA modules must be on the same LPAR that the IMS CTL region is on. These modules (ODBA) are loaded in the address space of the application, which is in turn loaded in the address space of the container. In this case, the container is WebSphere Application Server. As a result, the WebSphere Application Server installation must be on the same LPAR that the IMS DB itself is on. There is no isolation.

Figure 2-3 shows the architecture prior to IMS 11.

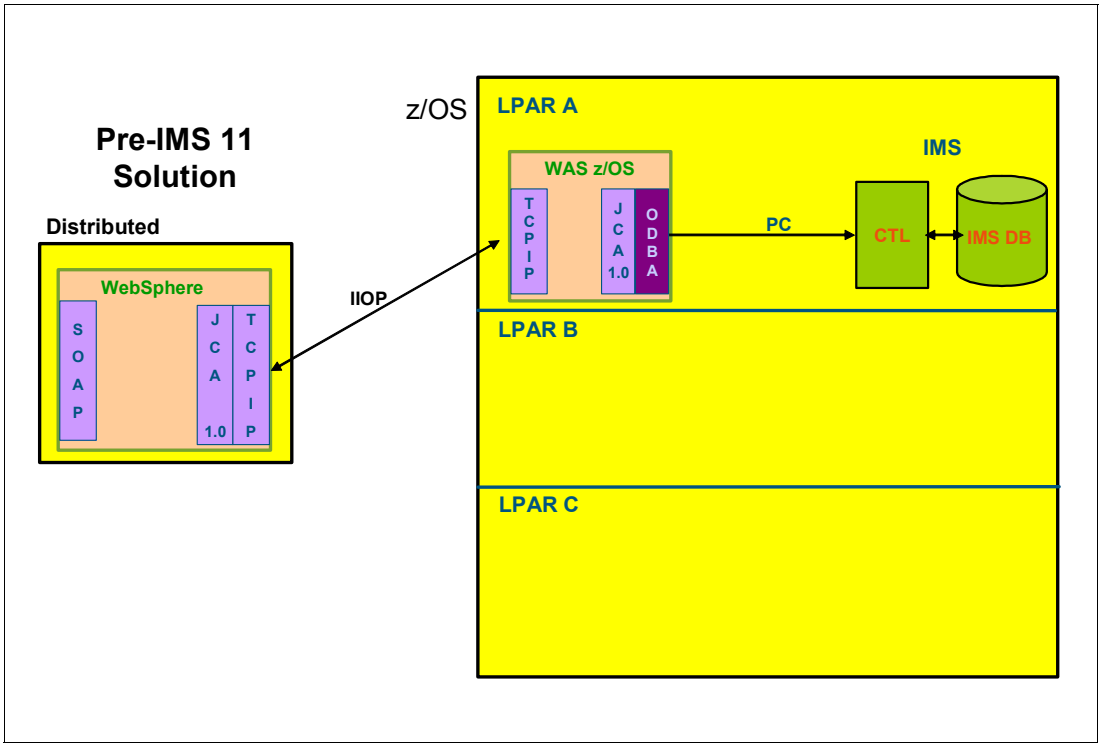


Figure 2-3 Architecture prior to IMS 11

By leveraging the Structured Call Interface (SCI), the applications can be on any LPAR in an IMSplex. SCI uses either PC or Cross-System Coupling Facility (XCF) calls to communicate with other SCI components. XCF allows calls to go across LPARs in an IMSplex, which allows applications (and their containers) to be isolated on their own LPARs.

Figure 2-4 on page 24 shows the effect of WebSphere Application Server leveraging the SCI and the cross-LPAR feature of IMS 11 Open Database Manager (ODBM).

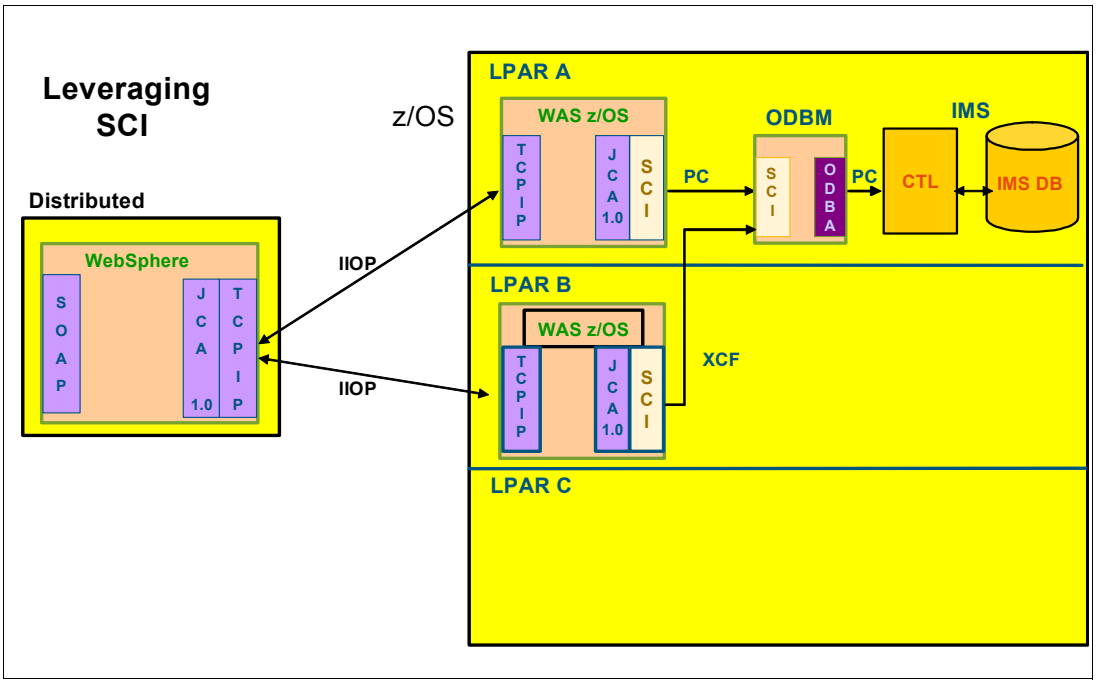


Figure 2-4 Effect of leveraging the SCI

IMS 11 has a Common Service Layer (CSL) address space, the IMS ODBM, to house the ODBA modules. This interface uses SCI as its communication mechanism. The ODBA modules are no longer tightly coupled with the applications themselves (and therefore the containers).

Figure 2-5 shows the Open Database environment.

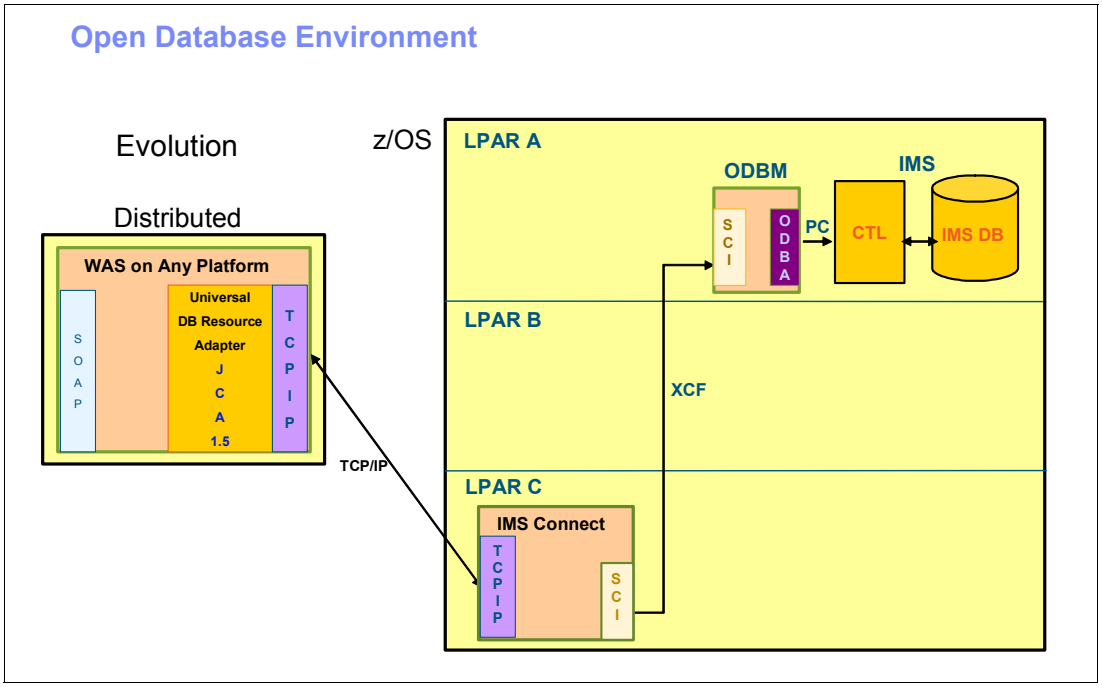


Figure 2-5 The Open Database environment with IMS Connect

2.4 IMS 11 architecture

This leads us to our real goal, which is to use IMS Connect as the complete gateway solution for IMS TM, OM, and now DB. IMS Connect is augmented to be an ODBM client, which allows distributed applications to use the Transmission Control Protocol/Internet Protocol (TCP/IP) to communicate with IMS Connect, which can then access any online database in the entire IMSplex.

Figure 2-6 shows the final architecture.

IMS Connect has become the IMS Gateway to both IMS TM and IMS DB. WebSphere and DB2 stored procedure no longer have to be on the same LPAR with IMS if they interface with the IMS ODBM address space. The ODBM address space must be on the same LPAR that IMS is on because of the use of the ODBA interface.

Distributed clients now have the option of going directly to IMS Connect for IMS DB requests. Existing DB Resource Adapter applications are unaffected by Open Database. To exploit Open Database from existing DB Resource Adapter applications, a migration to the JCA 1.5 programming model must occur.

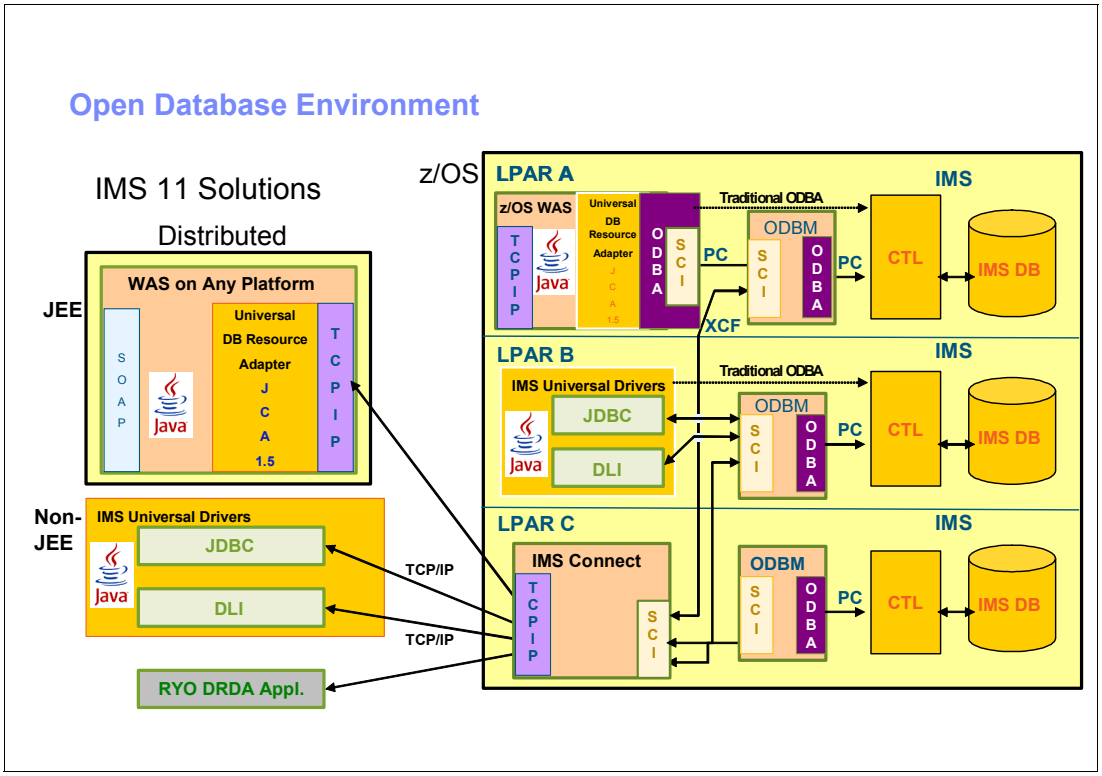


Figure 2-6 The final architecture

2.5 Open Database functions

In this section, we describe the following Open Database functions:

- IMS Open Database uses Distributed Relational Database Architecture
- Open Database Manager
- IMS Connect

- Distributed sync pointing
- Distributed Data Management

2.5.1 IMS Open Database uses Distributed Relational Database Architecture

Distributed Relational Database Architecture (DRDA) is an open vendor-independent architecture that provides a set of protocols and functions that allow connectivity between a client and database servers. DRDA is used for IMS DB access through TCP/IP, IMS Connect, and ODBM. DRDA provides:

- Communication protocol
- Two-Phase commit protocol
- Security

The structure is:

- IMS Connect acts as the application server or router for DRDA messages that are sent by way of TCP/IP protocol.
- ODBM routes the database connection requests that are received from IMS Connect to the IMS systems that are managing the requested database.
- The Distributed Data Management (DDM) architecture provides the command and reply structure for accessing distributed databases.
- ODBM translates the incoming database requests from the DDM protocol, which are submitted by the IMS-provided drivers and user-written DRDA applications, into the DL/I calls that IMS expects.

Manuals and further information about DRDA and DDM can be found at:

http://www.opengroup.org/dbiop

2.5.2 Open Database Manager

Open DataBase Manager is a new optional IMSplex Common Service Layer (CSL) component that runs in its own address space. ODBM uses the Structured Call Interface (SCI) services of the CSL for communications and Operations Manager (OM) services of the CSL for command processing. ODBM provides distributed and local access to IMS databases that are managed by IMS DB systems running in either the DBCTL or DB/TM environments in an IMSplex.

One ODBM instance must be defined in the IMSplex to use ODBM functions. Each z/OS image can have more than one ODBM. If multiple instances of ODBM are defined in the IMSplex, any ODBM instance can perform work from any z/OS image in the IMSplex.

ODBM routes the database connection requests that are received from IMS Connect to the IMS systems that are managing the requested database. Before establishing the connection to the IMS system, ODBM translates the incoming database requests from the DDM protocol, which are submitted by the IMS-provided connectors and user-written DRDA applications, into the DL/I calls that IMS expects. When ODBM returns the IMS output to the client, ODBM translates the response to the DDM protocol. From the ODBM perspective, application programs that interact directly with ODBM, such as IMS Connect, are ODBM clients. Users can create their own ODBM clients using the new ODBM CSLDMI API. ODBM client application programs can access databases that IMS DB manages on any LPAR in an IMSplex.

Independently and together with IMS Connect, ODBM supports various interfaces to ease the development of application programs that access IMS databases from many separate distributed and local environments. The supported ODBM interfaces are:

- IMS Universal DB resource adapter
- IMS Universal JDBC driver
- IMS Universal DL/I driver
- The ODBA interface
- The ODBM CSLDMI interface for user-written ODBM client application programs
- A a roll-your-own DRDA client

Figure 2-7 on page 28 shows an IMS configuration that includes ODBM.

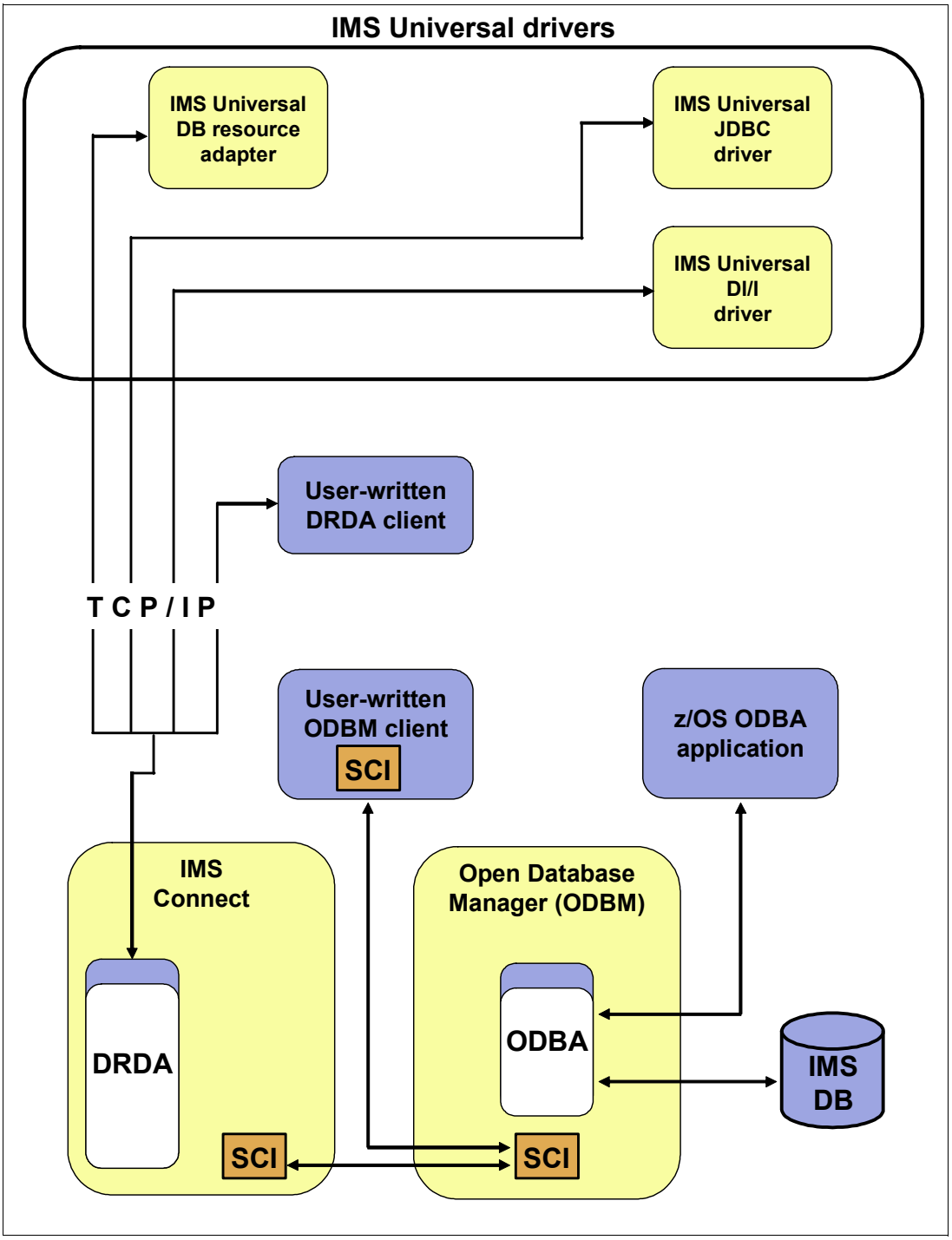


Figure 2-7 Overview of an IMS configuration that includes ODBM

Additionally as well as supporting the interfaces, ODBM provides the following functions:

- Supports 1PC (one phase commit) with or without RRS as the sync point coordinator. If the ODBM parameter RRS=N is specified, the CCTL interface using DRA is used to communicate with IMS instead of ODBA.

- Receives incoming database requests to IMS systems based on an alias name. Alias names for IMS systems are defined to ODBM on the CSLDCxxx PROCLIB member and can be specified on the incoming database requests by the client application programs:
  - If the client application program does not specify an alias name on a database access request, ODBM routes the request among multiple IMS systems by using a round-robin method of distribution, in which ODBM routes each incoming request. to the active IMS systems that are defined to ODBM.
- Before establishing the connection, ODBM translates the incoming database requests from the DDM protocol, which are submitted by the IMS-provided drivers and user-written DRDA applications, into the DL/I calls that IMS expects.
- Translates responses to the client into the DDM protocol.
- Enables ODBM client application programs to access databases from other LPARs within an IMSplex.
- Manages connections to ODBA.
- Protects IMS control regions from unexpected termination during DL/I processing of z/OS application programs that use the ODBA interface through ODBM.
- Functions as an RRMS resource manager and can participate in RRS-controlled sync point functioning for ODBA applications that are configured to use ODBM.
- Functions as a resource manager and issues the necessary calls to RRS for single-phase commit processing for local RRS transactions.
- Functions together with IMS Connect as a complete Distributed Relational Database Architecture (DRDA) target server for client application programs that use the DRDA specification.

Because ODBM and IMS Connect support DRDA, you can develop your own DRDA source application server that communicates with IMS using the Distributed Data Management (DDM) Architecture commands that are part of the DRDA specification.

From an ODBM perspective, application programs that interact directly with ODBM, such as IMS Connect, are ODBM clients. Users can create their own ODBM clients using the new ODBM CSLDMI API. ODBM client application programs can access databases that IMS DB manages on any LPAR in an IMSplex.

ODBM does not perform user authentication or authorization itself. If you are using IMS Connect with ODBM, you can authenticate the user IDs on request messages before they reach ODBM using IMS Connect security. In addition to being able to call RACF directly, IMS Connect provides the IMS DB Security user exit routine (HWSAUTH0) to facilitate customizing the security checking for communications with IMS DB.

You can have IMS check the authority of a user to allocate a PSB or access IMS resources using APSB and Resource Access Control (RAS) security.

If you are using RRS=Y in your ODBM, you have the option of using either APSB security with the ODBASE= parameter or Resource Access Security. If you have RRS=N, you can use RAS security.

To enable APSB security, specify the ODBASE parameter. RAS security is specified by the ISIS parameter. You can specify both the ODBASE and ISIS parameters on the IMS or DBC startup procedure or the DFSPBxxx PROCLIB member. If the ODBASE parameter is set to Y, whatever value is specified for the ISIS parameter is ignored.

When using APSB, you must:

1. Define the PSBs that you want RACF to protect to the AIMS or Axxxxxxx general resource class (where xxxxxxx is the value specified on the RCLASS= parameter of the IMS SECURITY macro).
2. Specify RCLASS=IMS | RACFCOM | RACFTERM | RASRACF | RAS on the IMS SECURITY macro at IMS system definition time.

When RAS is used, the actions you must perform depend on the value that is specified for the ISIS parameter. Table 2-1 shows the options for defining RAS security for applications that use ODBA.

Table 2-1 Options for defining RAS security for applications that use ODBA

Specifications	Actions to perform
ISIS=0 | N and ODBASE=N	No action required. No PSB security checking is performed.
ISIS=R and ODBASE=N	Define the PSBs that you want RACF to protect to the IIMS or Ixxxxxxx resource class, and then define the user IDs of the dependent region that you want authorized to access the PSBs. The ODBA support for IMS uses the security environment (ACEE) passed in the dependent region's task (TCBSENV), if present, or the dependent region's address space (ASXBSENV), if the ACEE is not present at the task leveL.
ISIS=C and ODBASE=N	Create a Resource Access Security exit routine that is named DFSRAS00. This routine must determine if the user is authorized to use the PSB.
ISIS=A and ODBASE=N	► Define the PSBs that you want RACF to protect to the IIMS or Ixxxxxxx resource class, and then define the user IDs of the dependent region that you want authorized to access the PSBs. The ODBA support for IMS uses the security environment (ACEE) that is passed in the dependent region's task (TCBSENV), if present, or the dependent region's address space (ASXBSENV), if the ACEE is not present at the task level. ► Create a Resource Access Security exit routine that is named DFSRAS00. This routine must determine if the user is authorized to use the PSB. RACF is called first, and then the exit routine is called.

If a user-written ODBM client passes a security object for a security product, such as RACF, ODBM invokes RACROUTE REQUEST=VERIFY to create an Accessor Environment Element for the APSB thread. IMS can then use the ACEE during APSB or RAS authorization for allocating PSBs or access to other resources.

2.5.3 IMS Connect

IMS Connect provides high performance TCP/IP and local z/OS communications between one or more IMS Connect clients and one or more IMS systems. IMS Connect, which previously provided access to only IMS Transaction Manager (IMS TM) through Open Transaction Manager Access (OTMA), now also provides access to IMS DB, in IMS 11, through ODBM. IMS Connect for Open Database enables distributed clients to exchange messages with IMS DB using TCP/IP connections and ODBM.

Communication with ODBM and OM requires IMS SCI.

With reference to WebSphere Application Server, WebSphere Application Server for z/OS can run in the same or a separate LPAR from IMS.

When located in a separate LPAR, it uses TCP/IP through IMS Connect:

- Distributed WebSphere Application Server does not require WebSphere Application Server for z/OS because communication is through IMS Connect.
- Distributed applications do not require a JEE (for example, WebSphere Application Server) environment.

IMS Connect enhancements for Open Database support

IMS Connect had the following enhancements in support of IMS Open Database:

- IMS Connect Configuration member HWSCFGxx:
  - New ODACCESS statement:
    - DRDA ports, timeout value, IMSplex name, and so on
- Changes to existing commands:
  - VIEWHWS, VIEWDS, VIEWPORT
- New Commands:
  - STARTOD, STOPOD, STARTIA, STOPIA, VIEWIA, SETOAUTO
- New User Exits:
  - HWSROUT0: Routing Exit for ODBM:
    - Can override the IMS alias and optionally select the ODBM target
  - HWSAUTH0: Security Exit for ODBM:
    - Can perform the authentication of the user ID

IMS Connect, with ODBM, supports the following types of clients in accessing IMS DB:

- Application programs using the IMS Universal DB resource adapter for the JEE platform
- Application programs using the IMS Universal JDBC driver
- Application programs using the IMS Universal DL/I driver
- User-written client application programs using the open standard DRDA communications architecture

For IMS Connect clients, such as the IMS Universal drivers, that access databases that are managed by IMS DB in DBCTL and DB/TM environment, IMS Connect manages TCP/IP connections and routes incoming access requests among the instances of the ODBM and the IMS DB systems in an IMSplex.

Because IMS Connect supports a subset of the DRDA protocol and, with ODBM, can be considered a DRDA target server, you can write application programs to the DRDA protocol directly; however, the IMS Universal DB resource adapter for the JEE platform is the recommended API for accessing IMS databases through TCP/IP from a distributed environment.

IMS Connect support for the IMS Universal drivers includes support for global two-phase commit transactions. IMS Connect supports communication with the IMS Universal drivers only on dedicated DRDA ports and only through shareable persistent sockets.

IMS Connect security support includes the IMS Connect DB Security user exit routine (HWSAUTH0), which can be used for greater control over the authentication of user IDs on connections that access IMS DB. RACF is also supported.

IMS Connect support for the IMS Universal drivers is defined by the ODACCESS configuration statement in the IMS Connect configuration PROCLIB member and requires at least one instance of ODBM running in the same IMSplex as IMS Connect.

Figure 2-8 shows an overview of IMS Connect support for IMS DB systems.

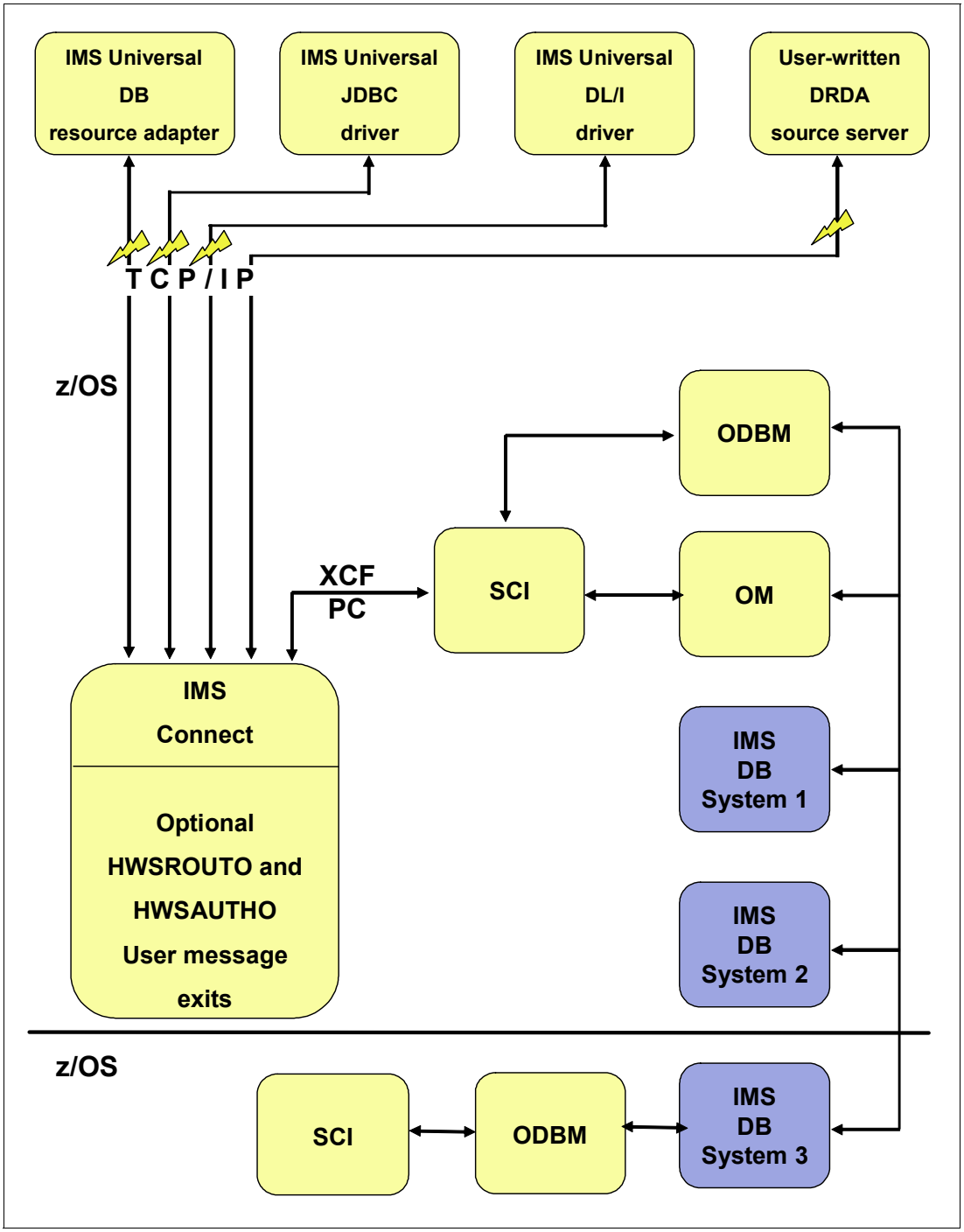


Figure 2-8 Overview of IMS Connect support for IMS DB systems

For clients that connect to IMS DB through ODBM, such as the IMS Universal drivers and clients using the Distributed Relational Database Architecture (DRDA), IMS Connect authenticates the user, but does not check the authority of the user to perform any actions.

To authenticate a user ID for an IMS DB client, IMS Connect can use the IMS Connect DB Security user exit routine (HWSAUTH0), a security product, such as RACF, or both:

- HWSAUTH0 user exit is always called by IMS Connect, regardless of whether RACF or another security product is enabled. If RACF support is included in your IMS Connect configuration, IMS Connect calls the HWSAUTH0 user exit before invoking RACF.
- HWSAUTH0 performs the authentication of the user ID/passticket of the ODBM client.
- HWSAUTH0 can override the input user ID with a separate user ID and can provide a RACF group ID to be authenticated further by IMS Connect.
- HWSAUTH0 is a BPE type-1 user exit and is refreshable.
- Password is passed in the clear.

RACF is enabled in IMS Connect for IMS DB and IMS TM clients by specifying RACF=Y in the IMS Connect configuration member or by issuing the IMS Connect command SETRACF ON.

IMS Connect does not support Secure Sockets Layer (SSL) directly for clients that connect to IMS DB. To secure connections to IMS DB with SSL, use IBM z/OS Communications Server Application Transparent Transport Layer Security feature (AT-TLS). The use of AT-TLS is transparent to IMS Connect.

IMS Connect provides several features to help you manage RACF passwords. Some of these features only apply when IMS Connect is configured to call RACF directly.

- When IMS Connect is configured to call RACF directly, users of the user message exit routines HWSSMPL0, HWSSMPL1, and HWSJAVA0 can change RACF passwords by submitting a client message that includes a password change request keyword.
- IMS Connect supports mixed-case passwords.
- An alternative to the RACF password is a PassTicket. PassTicket allows you to communicate with a host without using a RACF password. When IMS Connect is configured to call RACF directly, you can use PassTicket to authenticate user IDs and log on to computer systems that contain RACF.

For information about these features, refer to *IMS Version 11 Communications and Connections*, SC19-2433.

Configuring IMS Connect to call RACF: If you configure IMS Connect to call RACF, evaluate the impact of the RACF calls on IMS Connect performance.

IMS Connect routing and workload distribution

IMS Connect invokes the routing user exit first to allow the exit to select an ODBM and optionally override the IMS ALIAS:

- If the routing exit selects an ODBM, IMS Connect uses that ODBM and does not perform the round robin routing method.
- If the routing exit does not select an ODBM, IMS Connect selects an ODBM and performs the round robin routing method based upon the explicit ALIAS name and the blanked ALIAS name.
- If the routing exit overrides the IMS ALIAS, IMS Connect uses that IMS ALIAS.
- IMS Connect validates the ALIAS and the ODBM upon returning from the exit.

IMS Connect can also provide workload distribution by routing database connection requests to ODBM based on an alias name that is submitted by the client application program. To the

client application, the alias name represents the IMS system, or data store, to which the application program connects. Depending on the value of the alias name submitted, IMS Connect either routes the incoming connection request to a specific ODBM instance or distributes the incoming connection request to any available instance of ODBM in an IMSplex.

- ODBM clients can specify an IMS *ALIAS* in the message:
  - Alias represents the IMS data store that the client wants to send the message to:
    - Multiple Alias names for an IMS data store can be defined in the ODBM configuration member.
- If the client sends a message with a blank alias, IMS Connect routes the message to an ODBM using a round robin algorithm.
- If an alias points to multiple ODBMs, IMS Connect routes the message to one of those ODBMs using a round robin algorithm.

2.5.4 Distributed sync pointing

Figure 2-9 shows how Open Database achieves cross LPAR transaction management.

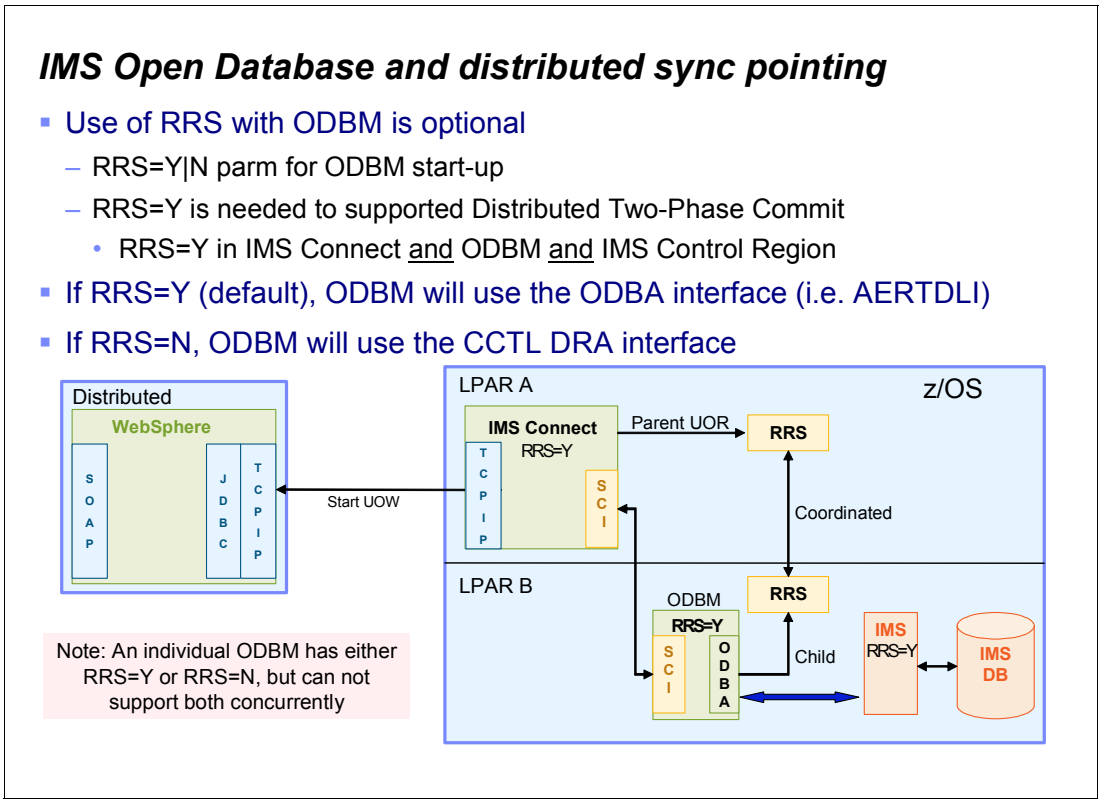


Figure 2-9 Open database cross LPAR transaction management

RRS=Y must be specified in the configuration for a global transaction.

When a client establishes a connection through IMS Connect to ODBM, several things are done to establish a coordinated Unit of Work:

1. IMS Connect creates the parent Unit of Recovery.
2. IMS Connect then sends the UR token to ODBM.
3. ODBM expresses interest in the UR as a child.

At this point we have a coordinated Unit of Recovery established.

For each request to access IMS data, connection information about the IMS host system, port number, and a valid user ID and password must be supplied to establish communication with IMS. A socket connection is first established to connect to the host IMS Connect system. When IMS Connect receives the request, it proceeds to authenticate the user based on the supplied user ID and password. After successful authentication, necessary information about the socket, such as PSB name and IMS alias (database subsystem) is sent to ODBA to allocate the PSB to connect to the database. An actual connection to an IMS database is only established when a PSB is allocated. Authorization for a particular PSB is done by the ODBA component during the allocation of a PSB.

Distributed Syncpoint (global transaction) requires RRS on z/OS.

Use of RRS with ODBM is optional. RRS=Y|N parameter for ODBM start-up:

- If RRS=Y (also the default), ODBM uses the ODBA interface (AERTDLI)
- If RRS=N, ODBM uses the DRA interface

Global transactions are not supported if RRS=N.

Distributed Relational Database Architecture (DRDA) is a set of protocols and functions that provide connectivity between a client and database servers:

- Communication protocol
- Two-Phase commit protocol
- Security

DRDA is used for IMS DB access through TCP/IP, IMS Connect, and ODBM.

IMS Connect serves as the TCP/IP server or router for DRDA messages sent by way of the TCP/IP protocol.

2.5.5 Distributed Data Management

IMS supports the Distributed Data Management (DDM) architecture of the Distributed Relational Database Architecture. You can develop your own source DDM server that communicates with the IMS target DDM server to provide access to databases that are managed by IMS DB in DBCTL and DB/TM IMS systems.

The IMS documentation for the DDM architecture includes only the DDM structures that are required to connect to and communicate with IMS and the DDM structures that IMS changed or defined. For the complete documentation about the DDM, see *DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture*, which is available from The Open Group at:

http://www.opengroup.org

The DDM architecture includes the following elements or terms:

- Commands
- Command objects
- Reply objects
- Reply messages

Each term, whether it is a command, command object, reply object, parameter, or message is represented by a code point. The code point is a hexadecimal value that represents and identifies the component in communication between a source server and the target server, for example:

- The EXCSAT command is represented by X'1041'
- The EXCSATRD reply object is represented by X'1443'
- The SRVNAM parameter is represented by X'116D', and so on

As an open standard, the DRDA specification requires that products that use the specification must conform to the conventions, protocols, standards, and so on, of its architecture. However, the DDM architecture that is a part of the DRDA specification allows products to create product-unique extensions, in which a product, such as IMS, uses a subset of the existing DDM-defined commands, parameters, messages, and product-unique structures that are defined by the product. When creating a product-unique extension that has product-unique structures, the product must conform to the DDM architecture.

The product-unique extension for IMS conforms to both the DDM architecture and the DRDA specification. IMS uses a subset of the existing DDM-defined commands, parameters, and messages, and a variety of IMS-defined structures that conform to the DDM architecture but are unique to IMS.

2.6 IMS 11 Universal drivers

Figure 2-10 shows the new IMS Universal drivers.

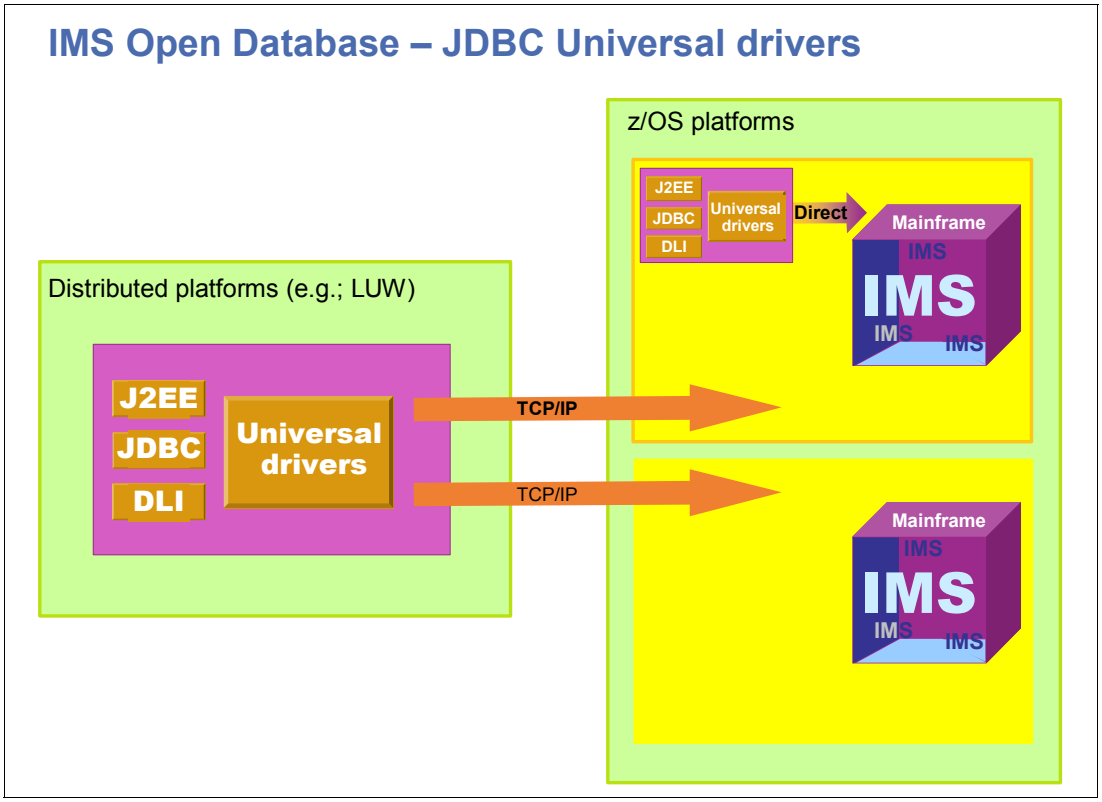


Figure 2-10 IMS Universal drivers

The Universal drivers have a framework that can process any of the three main programming models: JEE, JDBC, and DL/I. The Universal drivers can connect to any IMS subsystem on any mainframe system. The same application can have active connections to any number of IMS systems on any number of mainframe installations.

When running in a distributed environment on a server, such as WebSphere Application Server for distributed platforms, or in remote z/OS environments on a server, such as WebSphere Application Server for z/OS, the IMS Universal drivers connect to IMS using a type-4 connection architecture. This type of architecture supports TCP/IP communications and socket management.

When running locally on the same logical partition (LPAR) as IMS, the IMS Universal drivers connect to IMS using a type-2 connection architecture, which supports direct communication with IMS through the IMS Open Database Access and IMS database resource adapter interfaces.

WebSphere Application Server supports all of the IMS Universal drivers in both distributed and z/OS environments. CICS and DB2 for z/OS support the IMS Universal JDBC driver and the IMS Universal DL/I driver.

The IMS Universal drivers are software components that provide Java applications with direct, non-transactional connectivity and access to IMS databases from z/OS and distributed environments through TCP/IP. Applications using the IMS Universal drivers can reside on the same logical partition (LPAR) or on a separate LPAR from the IMS databases.

Programming approaches

The IMS Universal drivers provide an application programming framework that offers multiple options for access to IMS data. These programming options are:

- IMS Universal DB resource adapter

  Provides connectivity to IMS databases from a Java Platform, Enterprise Edition (Java EE) environment, and access to IMS data using the Common Client Interface (CCI) and Java Database Connectivity (JDBC) interfaces.

- IMS Universal JDBC driver

  Provides a stand-alone JDBC 3.0 driver for making SQL-based database calls to IMS databases.

- IMS Universal DL/I driver

  Provides a stand-alone Java API for writing granular queries to IMS databases using programming semantics that are similar to traditional DL/I calls.

Open standards

The IMS Universal drivers are built on the following industry open standards and interfaces:

- Java EE Connector Architecture (JCA)

  JCA is the Java standard for connecting Enterprise Information Systems (EISS), such as IMS, into the Java EE framework. Using JCA, you can simplify application development and take advantage of the services that can be provided by a JAVA EE application server, such as connection management, transaction management, and security management. The Common Client Interface (CCI) is the interface in JCA that provides access from JAVA EE clients, such as Enterprise JavaBean (EJB) applications, JavaServer Pages (JSP), and Java servlets, to back end IMS subsystems.

- Java Database Connectivity (JDBC)

  JDBC is the SQL-based standard interface for database access. It is the industry standard for database-independent connectivity between the Java programming language and any database that implemented the JDBC interface.

- Distributed Relational Database Architecture (DRDA) specification

  DRDA is an open architecture that enables communication between applications and database systems on disparate platforms. These applications and database systems can be provided by vendors and the platforms can be separate hardware and software architectures. DRDA provides distributed database access with built-in support for distributed, two-phase commit transactions.

- Distributed and local connectivity with the IMS Universal drivers

  The IMS Universal drivers support distributed (type-4) and local (type-2) connectivity to IMS databases.

Distributed connectivity with the IMS Universal drivers

With type-4 connectivity, the IMS Universal drivers can run on any platform that supports TCP/IP and a Java Virtual Machine (JVM), including z/OS. To access IMS databases using type-4 connectivity, the IMS Universal drivers first establish a TCP/IP-based socket connection to IMS Connect. IMS Connect is responsible for routing the request to the IMS databases using the Open Database Manager (ODBM), and sending the response back to the client application. The DRDA protocol is used internally in the implementation of the IMS Universal drivers. You do not need to know DRDA to use the IMS Universal drivers.

The IMS Universal drivers support two-phase commit (XA) transactions with type-4 connectivity. IMS Connect builds the necessary Recovery Resource Services (RRS) structure to support the two-phase commit protocol. If two-phase commit transactions are not used, RRS is not required.

Note: When establishing a connection to IMS, the driverType connection property must be set to indicate distributed (type-4) connectivity to IMS.

After successful authentication, the IMS Universal driver sends other socket connection information, such as program specification block (PSB) name and IMS database subsystem, to IMS Connect and ODBM to allocate the PSB to connect to the database. A connection to an IMS database is established only when a program specification block is allocated. Authorization for a particular PSB is done by the ODBM component during the allocation of a PSB.

The IMS Universal drivers support connection pooling with type-4 connectivity, which limits the time that is needed for allocation and deallocation of TCP/IP socket connections. To maximize connection reuse, only the socket attributes of a connection are pooled. These attributes include the IP address and port number that the host IMS Connect is listening on. As a result, the physical socket connection can be reused and additional attributes can be sent on this socket to connect to an IMS database. When a client application of the IMS Universal drivers using type-4 connectivity makes a connection to IMS, this means:

- A one-to-one relationship is established between a client socket and an allocated PSB that contains one or more IMS databases.
- A one-to-many relationship is established between IMS Connect and the possible number of database connections that it can handle at one time.
- IMS Connect does the user authentication.
- ODBM ensures that the authenticated user is authorized to access the given PSB.

You can also use the IMS Universal drivers with type-4 connectivity if your Java clients are running in a z/OS environment but are located on a separate logical partition from the IMS subsystem. Use type-4 connectivity from a z/OS environment if you want to isolate the application runtime environment from the IMS subsystem environment.

Local connectivity with the IMS Universal drivers

Local (or type-2) connectivity with the IMS Universal drivers is targeted for the z/OS platform and runtime environments. Use type-2 connectivity when connecting to IMS subsystems in the same logical partition (LPAR).

Note: Type-2 connectivity from WebSphere Application Server for z/OS and CICS environments is enabled by APAR PM13216.

Table 2-2 shows the z/OS runtime environments that support client applications of the IMS Universal drivers using type-2 connectivity.

Table 2-2 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity

z/OS Runtime Environment	IMS Universal Drivers with type-2 connectivity supported
WebSphere Application Server for z/OS	IMS Universal DB resource adapter
IMS Java-dependent regions (JMP and JBP regions) and CICS	IMS Universal DL/I driver IMS Universal JDBC driver

The DRDA protocol is not used to establish type-2 connectivity; instead, type-2 connectivity from WebSphere Application Server for z/OS is established using ODBA. Type-2 connectivity from a CICS environment uses the DRA.

Because it runs on the same LPAR that the IMS subsystem runs on, during connection time, a client application of the IMS Universal drivers using type-2 connectivity does not need to supply an IP address, port number, user ID, or password. The driverType property must be set to indicate local (type-2) connectivity to IMS. Type-2 support in JMP/JBP regions does not require TCP/IP or IMS Connect and uses the Java Native Interface to communicate directly with the CEETDLI interface to IMS DB. APAR PM02734 (PTF UK57317) enables the type-2 driver for all IMS V10 Java users.

RRSLocalOption connectivity type

In addition to type-4 and type-2 connectivity, the RRSLocalOption connectivity type is supported by the IMS Universal DB resource adapter that is running on WebSphere Application Server for z/OS. With RRSLocalOption connectivity, applications that use the IMS Universal DB resource adapter do not issue commit or rollback calls. Instead, transaction processing is managed by WebSphere Application Server for z/OS. The driverType property must be set to `RRSLocalOption` to enable this connectivity type.

Comparison of IMS Universal drivers programming approaches for accessing IMS

Depending on your IT infrastructure, solution architecture, and application design, choose the IMS Universal drivers programming approach that is best for your development scenario.

Table 2-3 on page 40 lists the recommended IMS Universal drivers' programming approach to use based on the application programmer's choice of application platform, data access method, and transaction processing option.

For standalone Java application (outside a Java EE application server) that resides on JMP and JBP regions:

- Use the IMS Universal JDBC driver (imsudb.jar), and make SQL calls with the JDBC API.
- Use the IMS Universal DL/I driver (imsudb.jar), and make DL/I calls with the PCB class.

The IMS Java dependent region resource adapter is a set of Java classes and interfaces that support IMS database access and IMS message queue processing within Java batch processing (JBP) and Java message processing (JMP) regions. The IMS Java dependent region resource adapter (imsutm.jar) provides Java application programs running in JMP or JBP regions with the same DL/I functionality that is provided in message processing program (MPP) and non-message driven BMP regions.

Table 2-3 Comparison of programming approaches

Application platform	Data access method	Transaction processing required	Recommended approach
WebSphere Application Server for distributed platforms or WebSphere Application Server for z/OS	CCI programming interface to perform SQL or DL/I data operations.	Local transaction processing only.	Use the IMS Universal DB resource adapter with local transaction support (imsudbLocal.rar), and make SQL calls with the SQLInteractionSpec class or DL/I calls with the DLIInteractionSpec class.
	CCI programming interface to perform SQL or DL/I data operations.	Two-phase (XA) commit processing* or local transaction processing.	Use the IMS Universal DB resource adapter with XA transaction support (imsudbXA.rar), and make SQL calls with the SQLInteractionSpec class or DL/I calls with the DLIInteractionSpec class.
	JDBC programming interface to perform SQL data operations.	Local transaction processing only.	Use the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter with local transaction support (imsudbJLocal.rar), and make SQL calls with the JDBC API.
	JDBC programming interface to perform SQL data operations.	Two-phase (XA) commit processing* or local transaction processing.	Use the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter with XA transaction support (imsudbJXA.rar), and make SQL calls with the JDBC API.

Application platform	Data access method	Transaction processing required	Recommended approach
Standalone Java application (outside a Java EE application server) that resides on a distributed platform or a z/OS platform	JDBC programming interface to perform SQL data operations.	Two-phase (XA) commit processing** or local transaction processing.	Use the IMS Universal JDBC driver (imsudb.jar), and make SQL calls with the JDBC API.
	Traditional DL/I programming semantics to perform data operations.	Two-phase (XA) commit processing** or local transaction processing.	Use the IMS Universal DL/I driver (imsudb.jar), and make DL/I calls with the PCB class.
	The IMS Java-dependent region resource adapter is a set of Java classes and interfaces that support IMS database access and IMS message queue processing within Java batch processing (JBP) and Java message processing (JMP) regions.	Two-phase (XA) commit processing** or local transaction processing.	Use the JDR resource adapter (imsutm.jar). It provides Java application programs running in JMP or JBP regions with the same DL/I functionality that is provided in message processing program (MPP) and non-message driven BMP regions.
Standalone non-Java application that resides on a distributed platform or a z/OS platform	Data access using DRDA protocol.	Two-phase (XA) commit processing or local transaction processing.	Use a programming language of your choice to issue DDM commands to IMS Connect. The application programmer is responsible for implementing the two-phase commit mechanism.

* XA transaction support is available only with type-4 connectivity.
** The driver is enabled for local and XA transactions, but the application programmer is responsible for implementing the two-phase commit mechanism. XA transaction support is available only with type-4 connectivity

Generating the runtime Java metadata class using the IMS Enterprise Suite DLIModel utility plug-in

To connect to an IMS database using the IMS Universal drivers or the IMS Classic drivers, you must include, on your Java class path, the Java metadata class that provides the database view.

The Java metadata class is a subclass of com.ibm.ims.db.DLIDatabaseView that is generated using the IMS Enterprise Suite DLIModel utility plug-in. The Java metadata class represents the application view information that is specified by a program specification block (PSB) and its related Program Control Blocks (PCBs). The Java metadata class provides a one-to-one mapping to the segments and fields that are defined in the PSB.

To generate the metadata class, use the DLIModel utility plug-in to import the application PSB source and related DBD source files. The Java metadata class must be compiled and made available through the class path for any Java application that is attempting to access IMS data using that PSB.

During database connection set up, pass the name of this metadata class to the Universal or Classic driver. The Java metadata class is used at runtime by the drivers to process both SQL and Java-based DL/I calls.

Chapter 4, “Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility” on page 77 provides the implementation details.

3

System environment

IMS Open Database uses several IMS components that all interface with each other when a Java application calls IMS to retrieve data. In this chapter, we detail the set up, configuration, and management of each component. In addition, we discuss how to secure access to IMS data when using the IMS Open Database capability. We cover these topics in the following sequence:

- Required environment setup for IMS Open Database
- Common Service Layer components:
  - Base Primitive Environment configuration
  - Structured Call Interface
  - Operations Manager
  - Open Database Manager:
    - ODBM commands
- IMS Connect:
  - First-time implementation: Set up and configuration
  - Modifying existing IMS Connect definitions for IMS Open DB Support
  - IMS Connect considerations for IMSplex
- Using IMS applications to help set up CSL and IMS Connect:
  - Installation Verification Program
  - Syntax Checker
- Security considerations

3.1 Required environment setup for IMS Open Database

For a Java application to access IMS data, it must send a request to IMS in a format that IMS can understand, and in turn, IMS must return the requested data in a format that the application can understand. The IMS Enterprise Suite DLIModel utility plug-in translates your existing IMS source data, consolidating it into a class file. Using this class file with the IMS Universal Drivers that are now available in IMS 11, a Java application can speak the language of IMS and retrieve the data using the IMS Open Database capability.

Note: We cover the installation, set up, and use of the IMS Enterprise Suite DLIModel utility plug-in separately in Chapter 4, “Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility” on page 77.

As for the flow of a Java application’s request for IMS data, it first sends a message to IMS Connect over TCP/IP using the DRDA protocol. IMS Connect then interfaces with the Open Database Manager (ODBM) using the Structured Call Interface (SCI). ODBM in turn uses SCI to route the request to the IMS that contains the requested data so that the data can be retrieved. The ODBM component is the key capability that enables this data access, whose configuration can be dynamically updated using Operations Manager (OM) commands that are known as type-2 commands, which we discuss later.

Important: The address spaces that are associated with the components that we mention here must be started in the following order:

1. SCI
2. OM
3. IMS control region
4. ODBM
5. IMS Connect

Working together, these components bridge the gap between the more modern Java development environment and the data that it requires, contained on the mainframe in IMS. We now explore how to set up, configure, and enable these components, beginning with the Common Service Layer.

For details about the architecture of IMS Open Database, see Chapter 2, “Open Database architecture” on page 19.

3.2 Common Service Layer components

The Common Service Layer (CSL) is an architecture that simplifies IMSplex management by providing a single point-of-control from a single system image, and plex-wide resource sharing. The Open Database capability introduces a new CSL address space called Open Database Manager that handles incoming database requests from distributed applications. Like other IMSplex members, the ODBM address space communicates with other members using SCI, which is another CSL component that we mentioned in the previous introduction section. The new ODBM address space can be queried and updated using type-2 commands, which are sent to IMS using the Operations Manager (OM) CSL component.

In this section, we cover setting up the CSL address spaces that are used with IMS Open Database: SCI, OM, and ODBM. Each of these are based on the Base Primitive Environment

(BPE); therefore, you must specify a BPE configuration member in each component's startup JCL.

Note: In addition to completing set up for each individual CSL address space, you must also define the name of your IMSplex in the DFSDFxxx PROCLIB member with the IMSPLEX parameter. Each CSL address space has an initialization member that also contains an IMSPLEX parameter, which must match the IMSplex name that you specified in the DFSDFxxx member. You can also specify this value in the DFSCGxxx PROCLIB member.

The xxx suffix of DFSDFxxx is specified on the CSLG startup parameter of IMS, so IMS knows which member contains the CSL definitions (either DFSDFxxx or DFSCGxxx). If you specify your CSL definitions in both of these PROCLIB members, DFSCGxxx takes precedence. For instructions about how to find a sample member definition, refer to the "OM, SCI, and DFSCGxxx samples" on page 68.

This section includes details about each of the address spaces that are required to use the IMS Open Database capability:

- Base Primitive Environment configuration
- Structured Call Interface
- Operations Manager
- Open Database Manager and commands used to dynamically manage it

3.2.1 Base Primitive Environment configuration

The Base Primitive Environment (BPE) configuration member enables tracing and defines the BPE execution environment values for running CSL address spaces and the IMS Connect address space. BPE trace records can be written to internal (memory only) trace tables and to external data sets. Use the TRCLEV parameter to indicate the type, level, and number of storage pages that are allocated (optional) for the trace table. The default setting is for the records to be written internally. The sample BPE configuration member for SCI and OM, named BPECONFG, takes this default in the bottom portion of Example 3-1. In Example 3-5 on page 48 and Example 3-7 on page 50, we reference the BPECONFG configuration member with the BPECFG= parameter (respectively) in our SCI and OM startup procedures.

Example 3-1 Sample BPE configuration member for SCI and OM address spaces

```
********************************************************************
* BPE CONFIGURATION FILE FOR SCI AND OM (BPECONFG)                  *
********************************************************************
LANG=ENU                              /* LANGUAGE FOR MESSAGES      */
                                      /* (ENU = U.S. ENGLISH)       */
#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,LOW,BPE)                    /* DEFAULT ALL TRACES TO LOW  */
# NOTE: KEEP THE FOLLOWING FOR COMPATIBILITY WITH 6.1 BPE
TRCLEV=(STG,LOW,BPE)                  /* STORAGE TRACE              */
TRCLEV=(CBS,LOW,BPE)                  /* CONTROL BLK SRVCS TRACE    */
TRCLEV=(DISP,LOW,BPE)                 /* DISPATCHER TRACE           */
TRCLEV=(AWE,LOW,BPE)                  /* AWE SERVER TRACE           */
TRCLEV=(LATC,LOW,BPE)                 /* LATCH TRACE                */
TRCLEV=(SSRV,LOW,BPE)                 /* SYSTEM SERVICES TRACE      */
TRCLEV=(USRX,LOW,BPE)                 /* USER EXIT SERVICES TRACE   */
```

```
#
# DEFINITIONS FOR OM/SCI TRACES
#
TRCLEV=(*,LOW,OM)                    /* DEFAULT OM TRACES TO LOW   */
TRCLEV=(*,LOW,SCI)                   /* DEFAULT SCI TRACES TO LOW  */
```

We chose to use a separate BPE configuration member named BPEODBM for initializing our ODBM address space, which Example 3-2 shows. Later, in Example 3-10 on page 54, you will see that we reference this configuration member with the BPECFG= parameter in our ODBM startup procedure.

Example 3-2 Sample BPE configuration member for ODBM address space

```
********************************************************************
* CONFIGURATION FILE FOR BPE FOR ODBM (BPEODBM)                    *
********************************************************************
LANG=ENU                             /* LANGUAGE FOR MESSAGES     */
                                     /* (ENU = U.S. ENGLISH)      */
#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,HIGH,BPE,PAGES=20)         /* DEFAULT TRACES TO HIGH    */
TRCLEV=(STG,MEDIUM,BPE)              /* STORAGE TRACE             */
TRCLEV=(CBS,MEDIUM,BPE)              /* CONTROL BLK SRVCS TRACE   */
TRCLEV=(DISP,HIGH,BPE)               /* DISPATCHER TRACE          */
TRCLEV=(AWE,HIGH,BPE)                /* AWE SERVER TRACE          */
TRCLEV=(SSRV,HIGH,BPE)               /* SYSTEM SERVICE TRACE      */
#-------------------------------------#
# DEFINITIONS FOR ODBM TRACES         #
#-------------------------------------#
TRCLEV=(*,HIGH,ODBM)                   /* SET DEFAULT FOR ALL ODBM   */
                                       /* TRACES TO HIGH.            */
```

Next, we include in Example 3-3 a final sample of a BPE configuration member named BPECFGIV that we later use when initializing IMS Connect. To see how this member is specified in our IMS Connect startup procedure, refer to Example 3-12 on page 64.

Example 3-3 Sample BPE configuration member for IMS Connect address space

```
********************************************************************
* CONFIGURATION FILE FOR BPE FOR IMS CONNECT (BPECFGIV)            *
********************************************************************
LANG=ENU                             /* LANGUAGE FOR MESSAGES     */
                                     /* (ENU = U.S. ENGLISH)      */
#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,HIGH,BPE,PAGES=20)         /* DEFAULT TRACES TO HIGH    */
TRCLEV=(STG,MEDIUM,BPE)              /* STORAGE TRACE             */
TRCLEV=(CBS,MEDIUM,BPE)              /* CONTROL BLK SRVCS TRACE   */
TRCLEV=(DISP,HIGH,BPE)               /* DISPATCHER TRACE          */
TRCLEV=(AWE,HIGH,BPE)                /* AWE SERVER TRACE          */
TRCLEV=(SSRV,HIGH,BPE)               /* SYSTEM SERVICE TRACE      */
#
# DEFINITIONS FOR IMS CONNECT TRACES
#
```

```
TRCLEV=(*,HIGH,HWS,PAGES=20)          /* DEFAULT TRACES TO HIGH    */
TRCLEV=(HWSI,HIGH,HWS,PAGES=100)      /* OTMA COMM ACTIVITY TRACE  */
TRCLEV=(HWSN,HIGH,HWS,PAGES=100)      /* LOCAL OPT DRIVER ACTIVITY */
TRCLEV=(HWSW,HIGH,HWS,PAGES=100)      /* TCP/IP DRIVER ACTIVITY    */
TRCLEV=(OTMA,HIGH,HWS,PAGES=100)      /* XCF CALLS TRACE           */
TRCLEV=(TCPI,HIGH,HWS,PAGES=100)      /* TCP/IP CALLS TRACE        */
```

Note: The IVP application contains a job that defines a sample BPE configuration member and adds it to IMS PROCLIB. Refer to “BPE, ODBM, and IMS Connect samples” on page 68, to determine how to locate this job within the IVP.

3.2.2 Structured Call Interface

Structured Call Interface (SCI) is the component of the CSL that enables IMSplex members to communicate with one another. Therefore, it is required that the SCI address space is started before any other IMSplex address spaces are started on a z/OS image. Each member must first register with SCI before it is allowed to join the IMSplex, even in the case of a single IMS system. We now discuss SCI implementation.

Configuration

The SCI initialization member CSLSIxxx must first be defined and configured. You must select a suffix to use with this member that you later reference on the SCI startup procedure. Refer to Example 3-4 for a sample configuration of the SCI initialization member, which we named CSLSI000.

Example 3-4 Sample configuration of the SCI initialization member

```
*-------------------------------------------------------------------*
* SCI INITIALIZATION PROCLIB MEMBER - CSLSI000                      *
*-------------------------------------------------------------------*
ARMRST=N,                     /* SHOULD ARM RESTART SCI ON FAILURE  */
SCINAME=SCI1,                 /* SCI NAME (SCIID = SCI1SC)          */
IMSPLEX(NAME=PLEXB)           /* IMSPLEX NAME (CSLPLEXB)            */
*-------------------------------------------------------------------*
* END OF MEMBER CSLSI000                                            *
*-------------------------------------------------------------------*
```

The parameters that you see defined in Example 3-4 are required. There is also an optional FORCE parameter that is associated with global interface storage that you can specify here, but it is not shown for simplicity. Let us examine our specifications in a bit more detail here:

- ARMRST=N: The SCI address space is not automatically restarted by the z/OS Automatic Restart Manager (ARM) capability in the event that a system abend occurs.
- SCINAME=SCI1: The SCI address space name SCI1 (can be 1-6 characters) is used to define an SCIID of SCI1SC for use in SCI processing.
- IMSPLEX(NAME=PLEXB): The IMSplex group name is PLEXB (can be 1-5 characters) and must match the IMSplex group name that is specified in the OM, ODBM, IMS initialization, and IMS Connect configuration members.

Note: The characters CSL are attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB.

There must be one SCI address space on each z/OS image where the CSL is active to allow communication with other components. These components can reside both within the same logical partition (LPAR) and on other z/OS images that are on other LPARs. For more information about configuring the SCI initialization member, refer to *IMS Version 11 System Definition*, GC19-2444.

Note: The IVP application contains a job that defines a sample SCI initialization member and adds it to IMS PROCLIB. Refer to “OM, SCI, and DFSCGxxx samples” on page 68 to determine how to find this job within the IVP.

Starting the SCI address space

To start the SCI address space as a task, execute the SCI procedure, and specify the xxx suffix of the configuration member CSLSIxxx on the SCIINIT= parameter. Example 3-5 shows a sample procedure that starts SCI as a task.

Example 3-5 Sample SCI startup procedure JCL

```
//*-------------------------------------------------------------------*
//*   SCI                                                             *
//*-------------------------------------------------------------------*
//*     PARAMETERS:                                                   *
//*     BPECFG  - NAME OF BPE MEMBER                                  *
//*     SCIINIT - SUFFIX FOR YOUR CSLSIXXX MEMBER                     *
//*     ARMRST  - INDICATES IF ARM SHOULD BE USED                     *
//*     SCINAME - NAME OF THE SCI BEING STARTED                       *
//*-------------------------------------------------------------------*
//*
//IEFPROC  EXEC PGM=BPEINI00,REGION=3000K,
//  PARM=('BPECFG=BPECONFG','BPEINIT=CSLSINI0','SCIINIT=000',
//            'ARMRST=N','SCINAME=SCI1')
//*
//STEPLIB  DD  DSN=IMS11B.SDFSRESL,DISP=SHR
//*
//PROCLIB  DD  DSN=IMS11B.PROCLIB,DISP=SHR
//*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//*
```

You must also designate the BPE configuration and initialization members that are associated with the SCI address space with the BPECFG and BPEINIT parameter values, respectively. We specified SCIINIT=000 in our sample, so the SCI initialization member CSLSI000 is read when the procedure is invoked. You can override certain parameter values that are defined in the CSLSI000 initialization member by specifying them here, for example, the ARMRST and SCINAME parameters that are specified in this member can override the values that are included in the SCI initialization member. Had the values that are shown in our sample procedure differed from those that are specified in CSLSI000, they prevail over those other settings when the procedure is invoked.

3.2.3 Operations Manager

OM is the component of the CSL that provides the single point-of-command entry into an IMSplex and is the focal point for operations management and automation. It uses SCI to communicate with other IMSplex members for command routing and for sending a

consolidated response to the command originator. The Open Database capability uses the ODBM address space to handle incoming IMS database requests. You can use the type-2 QUERY and UPDATE commands to display ODBM-related information and to update its configuration settings. You must use an OM interface when entering type-2 commands to one or more IMS systems that exist in an IMSplex. Therefore, we strongly recommend that you start an OM address space in your IMSplex (even if you only have a single IMS system) to enable this capability. Let us now discuss the steps that are required to implement OM.

Configuration

Just like the SCI component of the CSL, you must configure OM before you start its address space. Begin by defining the OM initialization member CSLOIxxx and defining its associated parameter values. We included a sample OM initialization member in Example 3-6, which is named CSLOI000.

Example 3-6 Sample configuration of the OM initialization member

```
*-------------------------------------------------------------------*
* OM INITIALIZATION PROCLIB MEMBER - CSLOI000                       *
*-------------------------------------------------------------------*
ARMRST=N,                      /* SHOULD ARM RESTART OM ON FAILURE  */
CMDLANG=ENU,                   /* USE ENGLISH FOR COMMMAND DESC     */
CMDSEC=N,                      /* NO COMMAND SECURITY               */
OMNAME=OM1,                    /* OM NAME (OMID = OM1OM)            */
IMSPLEX(
  NAME=PLEXB,                  /* IMSPLEX NAME (CSLPLEXB)           */
  AUDITLOG=SYSLOG.OM2Q01.LOG), /* MVS LOG STREAM                    */
CMDTEXTDSN=IMS11B.SDFSDATA     /* CMD TEXT DATASET                  */
*-------------------------------------------------------------------*
* END OF MEMBER CSLOI000                                            *
*-------------------------------------------------------------------*
```

Let us look at our initialization settings more closely:

- ARMRST=N: The z/OS Automatic Restart Manager (ARM) is not to restart the OM address space after an abend.
- CMDLANG=ENU: The English language is used for IMS command text that is distributed to OM automation clients upon request, which affects only the command descriptions that are displayed on a workstation SPOC that requests command text from OM.
- CMDSEC=N: We do not want any security checking to occur when a command is entered to the IMSplex from an OM API. Other settings can specify that RACF or an OM security user exit be used for command security.
- OMNAME=OM1: The OM address space name, OM1 (can be 1-6 characters), is used to define an OMID of OM1OM for use in OM processing.
- IMSPLEX(NAME=PLEXB): The IMSplex group name is PLEXB (can be 1-5 characters) and must match the IMSplex group name that is specified in the SCI, ODBM, IMS initialization, and IMS Connect configuration members. The characters CSL are attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB.

- IMSPLEX(AUDITLOG=SYSLOG.OM2Q01.LOG): The name of our z/OS system logger log stream that OM writes all IMSplex activity to, including command input/output and unsolicited messages (this function is optional and is not required to use the Open Database capability).
- CMDTXTDSN=IMS11B.SDFSDATA: The data set name for our PDS that contains the command syntax translatable text (can be 1-44 characters and must be a PDS with fixed length record members).

There must be one OM address space per IMSplex. You are not required to have an OM on on each z/OS image, which is the case with SCI. We do recommend that you have a backup OM in the IMSplex to be used in the event that the primary OM experiences a failure. For more information about configuring the OM initialization member, refer to *IMS Version 11 System Definition*, GC19-2444.

Note: The IVP application contains a job that defines a sample OM initialization member and adds it to IMS PROCLIB. Refer to “OM, SCI, and DFSCGxxx samples” on page 68 to determine how to find this job within the IVP.

Starting the OM address space

Use the OM startup procedure to start the OM address space as a task, specifying the xxx suffix of the OM initialization member CSLOIxxx to be used. Refer to Example 3-7 for a sample OM startup procedure.

Example 3-7 Sample OM startup procedure JCL

```
//*-------------------------------------------------------------------*
//*   OM                                                              *
//*-------------------------------------------------------------------*
//*     PARAMETERS:                                                   *
//*     BPECFG  - NAME OF BPE MEMBER                                  *
//*     OMINIT  - SUFFIX FOR YOUR CSLOIXXX MEMBER                     *
//*     ARMRST  - INDICATES IF ARM SHOULD BE USED                     *
//*     CMDLANG - LANGUAGE FOR COMMAND DESCRIPTION TEXT               *
//*     CMDSEC  - COMMAND SECURITY METHOD                             *
//*     OMNAME  - NAME OF THE OM BEING STARTED                        *
//*-------------------------------------------------------------------*
//*
//IEFPROC  EXEC PGM=BPEINI00,REGION=3000K,
//  PARM=('BPECFG=BPECONFG','BPEINIT=CSLOINIO','OMINIT=000',
//        'ARMRST=N','CMDSEC=N','OMNAME=OM1')
//*
//STEPLIB  DD  DSN=IMS11B.SDFSRESL,DISP=SHR
//*
//PROCLIB  DD  DSN=IMS11B.PROCLIB,DISP=SHR
//*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//*
```

Because OM is a CSL address space and it based on BPE, you must specify the BPE configuration and OM initialization members to be used, for example, we specified OMINIT=000, so we know that the OM initialization member CSLOI000 will be read when we start this procedure. The ARMRST, CMDSEC, and OMNAME parameter values can all be

specified in this procedure to override the definitions that are included in the CSLOI000 initialization member.

3.2.4 Open Database Manager

The Open Database capability uses the ODBM to handle IMS database access requests from distributed applications and z/OS applications. It uses SCI for communication and OM for command processing capability. We now review the steps that are required to define and enable the ODBM component.

Configuration

One ODBM instance must be defined in the IMSplex to use ODBM functions. Each LPAR that contains an IMS control region must have at least one ODBM address space but is not limited to that single instance. If multiple instances of ODBM are defined in the IMSplex, any of them can fulfill requests that are received from other z/OS images in the IMSplex. Two OBDM-related PROCLIB members must be defined before starting the ODBM address space: CSLDIxxx and CSLDCxx, which we now review.

CSLDIxxx

Like the other CSL components, each ODBM has an initialization member named CSLDIxxx which must be defined, similar to our sample member CSLDI001, which Example 3-8 shows.

Example 3-8 Sample ODBM initialization member

```
*********************************************************************
**  CSLDI001 member:                                               *
**  This PROCLIB member is specified by the ODBMINIT=001           *
**  value on the ODBM start up procedure.                          *
**                                                                 *
**  Parameters specified here are used for ODBM initialization.    *
**                                                                 *
**  ODBM configuration parameters are specified in the             *
**  CSLDC001 PROCLIB member which can be specified by either       *
**  the ODBMCFG=001 EXEC parameter or in this PROCLIB member       *
**  on the ODBMCFG=001 parameter.                                  *
**                                                                **
*********************************************************************
ODBMNAME=IMSBBO
IMSPLEX(NAME=PLEXB)
ODBMCFG=001
```

Looking more closely at our parameter definitions:

- ODBMNAME=IMSBBO: The name of our ODBM address space, IMSBBO (can be 1-6 characters), is used to define an ODBMID of IMSBBOOD for use in ODBM processing. If you have additional ODBM address space instances in your IMSplex, you must define a separate CSLDIxxx initialization member for each one and ensure that a unique name is used (or designate a unique name on the startup procedure).
- IMSPLEX(NAME=PLEXB): The IMSplex group name is PLEXB (can be 1-5 characters) and must match the IMSplex group name that is specified in the SCI, OM, IMS initialization, and IMS Connect configuration members. The characters CSL are attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB.
- ODBMCFG=001: By specifying a suffix of 001, we set the name of our ODBM configuration member to be CSLDC001, which contains definitions for our Open Database

Access (ODBA) connection initialization parameters and additional ODBM configuration statements, which we explore in the following section

You can also specify the ARMRST parameter in the ODBM initialization member, which indicates whether the z/OS Automatic Restart capability automatically restarts the ODBM address space in the event of a failure.

Also not shown in our example is the RRS= parameter, which indicates whether ODBM uses z/OS Resource Recovery Services (RRS). While it is optional to specify this keyword, ODBM runs with RRS by default, which is significant because in this case, both IMS Connect and the IMS control region must also run with RRS.

Note: RRS is the z/OS sync point manager and is responsible for coordinating all of the resource managers that are associated with a Unit Of Recovery (UOR). When ODBM runs with RRS, it uses a sync point protocol called two-phase commit (2PC), which ensures that either *all* or *none* of an application program's resource updates are made to a set of resources. Refer to *IMS Version 11 Application Programming*, SC19-2428 for more detail about 2PC sync point protocol.

ODBM is required to run with RRS when ODBA access is needed, which is the case when IMS data is requested by DB2 stored procedures or from an application that is running in WebSphere Application Server for z/OS. When ODBM runs without RRS, it functions like a Coordinator Controller (CCTL) that connects to IMS data through a database resource adapter (DRA) interface.

Refer to the manual *IMS Version 11 System Definition,* GC19-2444, for details about the CSLDIxxx member.

Tip: You can find a sample job that defines an ODBM initialization member and adds it to IMS PROCLIB within the IVP application. Refer to "BPE, ODBM, and IMS Connect samples" on page 68, for more details.

Now that we defined our ODBM initialization member to be used in starting the ODBM address space, we now must set up the ODBM configuration member, CSLDCxxx. Defining this member connects ODBM to one or more IMS systems, which are referred to as data stores in this arena. The ODBMCFG parameter that is defined in the ODBM CSLDIxxx initialization points to this configuration member, which we now review.

CSLDCxxx

ODBM must be configured to recognize the IMS data stores that are referenced as alias names by incoming IMS database requests from application programs. The CSLDCxxx member establishes these associations and contains a global section with settings that apply to all IMS data stores and a local section with settings that apply to specific data stores (which override the global setting if specified). In the previous section, we specified ODBMCFG=001 in our CSLDI001 ODBM initialization member, indicating that CSLDC001 is used for our ODBM configuration member. We show a sample of this member in Example 3-9.

Example 3-9 Sample ODBM configuration member

```
*********************************************************************
**  This PROCLIB member is specified by the ODBMCFG=001            **
**  value on the ODBM start up procedure or from the               **
**  ODBMCFG=001 parameter in the DFSDI001 PROCLIB member           **
**                                                                 **
**  Parameters specified here are used for ODBM configuration.     **
```

```
**                                                                    **
**  This member is split into 2 sections.                             **
**                                                                    **
**  <SECTION=GLOBAL_DATASTORE_CONFIGURATION>                          **
**  <SECTION=LOCAL_DATASTORE_CONFIGURATION>                           **
**                                                                    **
**********************************************************************
**  <SECTION=GLOBAL_DATASTORE_CONFIGURATION>                          **
**                                                                    **
**  This section defines configuration parameters that will be        **
**  used when their corresponding parameters are not present in       **
**  the local data store section.                                     **
**                                                                    **
**  Defaults shown                                                    **
**  IDRETRY=0  (Can only be specified here.)                          **
**  TIMER=60   (Can only be specified here.)                          **
**  MAXTHRDS=1                                                        **
**  FPBUF=000                                                         **
**  FPBOF=000                                                         **
CNBA=325
**********************************************************************
**  <SECTION=LOCAL_DATASTORE_CONFIGURATION>                           **
**                                                                    **
**  This section defines ODBM configuration parameters.               **
**                                                                    **
**  Multiple ODBMs may be appended, each with its unique              **
**  ODBMNAME=odbmname. Optional parameters that are not               **
**  specified in an ODBM configuration statement group will           **
**  take on the corresponding parameter value specified in            **
**  the global data store section, if one was specified.              **
**                                                                    **
**  ODBM(NAME=odbmname,                                               **
**       DATASTORE(NAME=datastorename),                               **
**       ALIAS(NAME=aliasname),                                       **
**       FPBUF=nnn,                                                   **
**       FPBOF=nnn,                                                   **
**       CNBA=nnn,                                                    **
**       MAXTHRDS=nnn                                                 **
**                   )                                                **
**                                                                    **
**  MAXTHRDS=                                                         **
**  FPBUF=                                                            **
**  FPBOF=                                                            **
**  CNBA=                                                             **
**********************************************************************
<SECTION=LOCAL_DATASTORE_CONFIGURATION>
ODBM(NAME=IMSBBO
DATASTORE(NAME=IMSB
ALIAS(NAME=IMS2)
)
)
```

The comments at the beginning of the sample indicate which defaults are used for all ODBM instances, if the parameter settings are not specified. It also indicates which sections the parameters can be specified in—either global, local, or both. In our local section, we establish

the association between our ODBM instance and a specific IMS control region or data store. Multiple ODBMs can be defined here and each one's associated data store can have multiple alias names using the ALIAS parameter. Parameters that are not defined in the local section default to the values that are defined in the global section.

We specified our ODBM name to be IMSBBO and created an association with the IMSB control region using the DATASTORE statement. An application program does not know the name of the data store (IMSID or control region name) and can only reference it as an alias name. The IMSB data store can be referenced as IMS2 by incoming application program requests for IMS data because we specified an alias name of IMS2 using the ALIAS statement. The alias name can be different than the data store name. Specify these settings in the local section of the configuration member, the start of which is designated by the <SECTION=LOCAL_DATASTORE_CONFIGURATION> header.

ALIAS parameter: The ALIAS parameter is optional. If one is not specified in this section, an application can reference the data store name and IMS internally creates an ALIAS name using the data store name.

There are certain parameters that you can only specify in the global section, which begins with the <SECTION=GLOBAL_DATASTORE_CONFIGURATION> header. Specifically, you can use the IDRETRY parameter to specify the number of attempts an ODBM instance must make to connect to a data store when it does not connect right away, and use the TIMER parameter to indicate how many seconds must elapse before the connection is attempted again. As you can see, we accepted the defaults in our sample.

You can define the remaining parameters in the configuration member in either the local or the global section. As a reminder, any parameter that is defined in the local section overrides any definitions in the global section. In Example 3-9 on page 52, we only defined the CNBA global parameter (specifies the total number of Fast Path NBA buffers for ODBM use), whereas we have taken the default setting for each of the others. Use the MAXTHRDS parameter to indicate how many concurrently active threads an IMS data store can have. Use the FPBUF and FPBOF parameters to show how many Fast Path DEDB buffers and Fast Path DEDB overflow buffers are allocated per thread, respectively. You can find more information about how ODBM configuration parameters are defined in CSLDCxxx in the manual *IMS Version 11 System Definition,* GC19-2444.

Tip: You can find a sample job that defines an ODBM configuration member and adds it to IMS PROCLIB within the IVP application. Refer to "BPE, ODBM, and IMS Connect samples" on page 68 for more detail.

Starting the ODBM address space

Use the ODBM startup procedure to start the ODBM address space as a task, specifying the BPE configuration member and suffix of the ODBM initialization member to be used. Keep in mind that startup parameters that are specified here in the procedure override those that are predefined in our CSLDI001 ODBM initialization member. Refer to Example 3-10 for a sample of the ODBM startup procedure that we used.

Example 3-10 Sample ODBM startup procedure JCL

```
//*****************************************************************
//*     ODBM Procedure
//*
//*
//*     Parameters:
//*     BPECFG  - Name of BPE member
```

```
//*     BPEINIT  - CSLDINI0, the module that contains the ODBM start up values
//*     ODBMINIT  - Suffix for your CSLDIxxx member
//*     PARM1   - other override parameters:
//*               ARMRST  - Indicates if ARM should be used
//*               ODBMNAME  - Name of ODBM being started
//*               ODBMCFG  - Suffix for your CSLDCxxx member
//*               RRS     - Indicates RRS is (Y) or is not (N) used
//*
//*               example:
//*               PARM1='ARMRST=Y,ODBMNAME=ODBM1,ODBMCFG=000,RRS=N'
//*
//IMSBBOPD  PROC RGN=0M,TME=1440,SOUT=*,
//             BPECFG=BPEODBM,
//             ODBMINIT=001,
//             CSLDI=CSLDINI0,
//             PARM1='RRS=Y,ARMRST=N'
//*
//ODBMPROC EXEC PGM=BPEINI00,REGION=&RGN,TIME=&TME,
//  PARM='BPECFG=&BPECFG,BPEINIT=&CSLDI,ODBMINIT=&ODBMINIT,&PARM1'
//STEPLIB  DD   DSN=IMS11B.SDFSRESL,DISP=SHR
//         DD   DSN=SYS1.CSSLIB,UNIT=SYSALLDA,DISP=SHR
//PROCLIB  DD   DSN=IMS11B.PROCLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=&SOUT
//SYSUDUMP DD   SYSOUT=&SOUT
```

We designated BPECFG=BPEODBM, so we know that the BPEODBM member that is defined in Example 3-2 on page 46 will be used in starting the ODBM address space. We also specified ODBMINIT=001, so the ODBM initialization member, CSLDI001, which is shown in Example 3-8 on page 51, will be read when we start this procedure. The BPE initialization member that will be used is CSLDINI0 (not shown in this book). The ARMRST and RRS parameter values can both be included in this procedure to override the definitions that are contained in the CSLDI001 initialization member.

ODBM user exits

You are able to customize and monitor your ODBM environment by writing one or more of the following ODBM user exits, called and managed by BPE:

- CSL ODBM Initialization and Termination user exit: Called when an ODBM or IMSplex initializes or terminates. It can be used to track the timing of these events.
- CSL ODBM Input user exit routine: Called when a CSLDMI request (from the ODBM API) is issued. It can change segment search arguments (SSAs), an I/O area, or an application interface block (AIB).
- CSL ODBM Output user exit routine: Called when an ODBM is returning a response to an ODBM client that sent in a request. It can view and modify response data before sending it to the client.
- CSL ODBM Client Connect and Disconnect user exit routine: Called when an ODBM client registers to or de-registers from ODBM.
- CSL ODBM statistics available through BPE statistics user exit: Tracks statistics about both BPE and ODBM.

For additional details about any of these exits, refer to the manual *IMS Version 11 Exit Routines*, SC19-2437.

ODBM commands

You can issue type-2 commands to display ODBM information with the QUERY ODBM command and update the configuration settings with the UPDATE ODBM command. These are type-2 commands, so you must enter them through an OM interface, such as the IMS TSO SPOC application (also referred to as just TSO SPOC), which we show several examples of shortly. Using this application, you can send these commands to the entire IMSplex. To invoke the TSO SPOC, select option 1 from the IMS application menu, shown in Figure 3-9 on page 66.

Note: Before you can issue IMS commands from the TSO SPOC application, you must set your desired user preferences from the OPTIONS menu.

QUERY ODBM

You can display information about ODBM in several aspects, including:

- Alias names that are associated with all ODBM instances in the IMSplex. Figure 3-1 shows sample output from the QUERY ODBM TYPE(ALIAS) command, which shows that two alias names IMS2 and IMSZ are associated with our ODBM instance.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(ALIAS) SHOW(ALL)
MbrName  AliasName   CC AliasStatus ThreadCount DatastoreName DatastoreStatus
IMSBBOOD IMSZ         0 STARTED               0 IMSB          STARTED
IMSBBOOD IMS2         0 STARTED               1 IMSB          STARTED









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-1 Sample output from QUERY ODBM TYPE(ALIAS) SHOW(ALL) command

- Configuration information for all ODBM instances drawn from each one's CSLDCxxx configuration member. See Figure 3-2 on page 57 for sample output from the QUERY ODBM TYPE(CONFIG) command, which shows the parameter definitions. Scrolling to the right displays two more columns named Timer and Aliases, which are not shown here. For this command, the output is sorted by configuration member name.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(CONFIG) SHOW(ALL)                  More:    >
MbrName  ConfigName Global DatastoreName   CC FPBUF FPBOF CNBA MaxThrds IdRetry
IMSBBOOD CSLDC001   Y                       0
IMSBBOOD CSLDC001   N      IMSB             0                        50









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-2 Sample output from QUERY ODBM TYPE(CONFIG) SHOW(ALL) command

- Configuration information for ODBM drawn from its CSLDCxxx configuration member, displayed by data store. Figure 3-3 shows sample output from the QUERY ODBM TYPE(DATASTORE) command and shows how we defined the parameter values. Scrolling to the right displays two more columns: the CNBA, MaxThrds, IdRetry, and Timer values, which are not shown here. For this command, the output is sorted by data store name.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(DATASTORE) SHOW(ALL)               More:    >
MbrName  DatastoreName   CC ConnectionStatus ThreadCount Aliases   FPBUF FPBOF
IMSBBOOD IMSB             0 STARTED                    0 IMS2,IMSZ     0     0









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-3 Sample output from QUERY ODBM TYPE(DATASTORE) SHOW(ALL) command

- Clients of ODBM, as shown in the sample output for the QUERY ODBM TYPE(SCIMEMBER) command in Figure 3-4 on page 58. The command response shows one ODBM client, IMS Connect.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(SCIMEMBER) SHOW(ALL)
MbrName  SCIMbrName   CC Type     ThreadCount
IMSBBOOD IMSBBHWS      0 IMSCON             0









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-4 Sample output from QUERY ODBM TYPE(SCIMEMBER) SHOW(ALL) command

- Individual ODBM threads. You can filter the command response to show only those threads that are associated with certain specified PSBs, ODBM clients, such as IMS Connect, data stores, and threads that have a specific status. Refer to the sample response to the QUERY ODBM TYPE(THREAD) command in Figure 3-5 on page 59. We did not include a filter for the command. Instead we chose to display the IMS Connect name and PSB that are associated with the ODBM thread that is currently running. We scrolled to the right to display the information of interest.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(THREAD) SHOW(PSB SCIMEMBER)          More: <
MbrName  ThreadID                          AliasName SCIMbrName PSB
IMSBBOOD 255388382556ACE015A1534004619040 IMS2      IMSBBHWS   AUTPSB11









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-5 Output from QUERY ODBM TYPE(THREAD) SHOW(PSB SCIMEMBER) command

- Trace status, which you can see an example of in the response to the QUERY ODBM TYPE(TRACE) command shown in Figure 3-6. We currently do not have an active trace running for our ODBM instance.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: QUERY ODBM TYPE(TRACE) SHOW(ALL)
MbrName  DatastoreName   CC TraceStatus
IMSBBOOD IMSB             0 INACTIVE









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Retrieve F12=Cancel
```

Figure 3-6 Sample output from QUERY ODBM TYPE(TRACE) SHOW(ALL) command

UPDATE ODBM

You can dynamically change ODBM configuration settings with the type-2 UPDATE ODBM command. You can also use this command to start and stop data stores, aliases, and tracing activity. We now discuss each type of UPDATE ODBM command in detail:

- If you need to dynamically stop a connection between ODBM and IMS, you can do so with the UPDATE ODBM STOP(CONNECTION) command. On this command, you can specify one or more data stores or alias names, as shown in Figure 3-7. When the connection is broken, you will also see a message issued to the system console similar to: CSL4009I ODBM Disconnected from IMS data store IMSB IMSBBOOD.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control             Keylist saved
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: UPDATE ODBM STOP(CONNECTION) DATASTORE(IMSB)
MbrName  DatastoreName   CC
IMSBBOOD IMSB             0









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Swap     F12=Cancel
```

Figure 3-7 Sample response for UPDATE ODBM STOP(CONNECTION) DATASTORE(IMSB)

You can dynamically start the connection between ODBM and IMS again with the UPDATE ODBM START(CONNECTION) command. See Figure 3-8 on page 61 for a sample. When the connection is established, you will also see a message issued to the system console that is similar to: CSL4004I ODBM Connected to IMS data store IMSB IMSBBOOD.

Note: You can start and stop connections between OBDM and a data store or between ODBM and an alias. These two are mutually exclusive and it is important to understand that when you start or stop a connection between ODBM and an alias, it does not impact ODBM's connection to the actual data store. Therefore, it is possible to stop an ODBM connection to an alias, but the ODBM will still be connected to the associated data store afterwards. The ability to start and stop a connection to an alias is useful because it allows you to isolate programs using a specific alias, and other aliases that are still connected to ODBM can still access the underlying data store.

```
 File  Action  Manage resources  SPOC  View  Options  Help
 sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
 PLEXB                    IMS Single Point of Control
 Command ===>

  sssssssssssssssssssss   Plex . .        Route . .            Wait . .
 Response for: UPDATE ODBM START(CONNECTION) DATASTORE(IMSB)
MbrName  DatastoreName   CC
IMSBBOOD IMSB             0









  F1=Help      F3=Exit      F4=Showlog   F6=Expand    F9=Swap     F12=Cancel
```

Figure 3-8 Sample response for UPDATE ODBM START(CONNECTION) DATASTORE(IMSB)

Using the UPDATE ODBM commands you can also dynamically change the ODBM's configuration. To update the configuration, use the commands in the following sequence:

```
UPDATE ODBM STOP(CONNECTION) DATASTORE(*)
UPDATE ODBM TYPE(CONFIG) MEMBER(xxx)
UPDATE ODBM START(CONNECTION) DATASTORE(*)
```

This sequence reads and activates PROCLIB member CSLDCxxx. If you change the suffix xxx, you must also remember to change member CSLDIxxx or the ODBM task JCL; otherwise, ODBM reverts to the old settings when it is next started.

Note: As you can infer from the command sequence example, ODBM requires the existing data stores to be stopped before the configuration can be changed.

You can find more command examples in the manual *IMS Version 11 Commands, Volume 2: IMS Commands N-V*, SC19-2431.

3.3 IMS Connect

IMS Connect uses ODBM to enable distributed DRDA application access to IMS data. You might already be using IMS Connect for TCP/IP access to IMS transactions or commands.

But if you have not yet implemented IMS Connect, you must implement it to use the Open Database capability. This section examines both scenarios.

Important: The user ID that is associated with a request for IMS data from a distributed application must be authenticated in IMS Connect. Refer to 3.5, “Security considerations” on page 73 for recommendations and further detail about how to accomplish this.

3.3.1 First-time implementation: Set up and configuration

If you are already using IMS Connect, go to “IMS Connect user exits” on page 64. Otherwise, begin by defining two configuration members: one for BPE and one for IMS Connect. After these members are in place, you can start the IMS Connect address space as a task.

BPECFxxx

First, we define our IMS Connect BPE execution environment settings in the BPECFxxx member. This member contains IMS Connect message language information and tracing specifications for IMS Connect’s internal trace tables. Refer to Example 3-3 on page 46 for a sample of the member named BPECFGIV that we use to initialize our IMS Connect address space.

HWSCFxxx

Next, you must define your IMS configuration member to designate IMS Connect’s environment settings to establish how IMS Connect will communicate with TCP/IP, the CSL and ODBM, among other components. You must define several parameters in this member, which are very thoroughly documented in the manual *IMS Version 11 System Definition*, GC19-2444 and in the book *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity,* SG24-6794. Let us now take a look at the way we defined our IMS Connect configuration member, HWSCFODB, in Example 3-11.

Example 3-11 Sample IMS Connect configuration member

```
*********************************************************************
* CONFIGURATION FILE FOR IMS CONNECT (HWSCFODB)                      *
*********************************************************************
HWS=(ID=IMSBBHWS,XIBAREA=100,RACF=N,RRS=Y)
TCPIP=(HOSTNAME=TCPIP)
ODACCESS=(ODBMAUTOCONN=Y,IMSPLEX=(MEMBER=IMSBBHWS,TMEMBER=PLEXB),
     DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000)
```

In Example 3-11, we defined several parameters:

- HWS=:
  - (ID=IMSBBHWS): The name of our IMS Connect address space is IMSBBHWS.
  - (XIBAREA=100): We allow 100 full words to be allocated for the XIB user area.
  - (RACF=N): The user ID is passed to IMS without calling RACF for authentication.

    Note: If no user ID is passed by the calling application, IMS Connect does not set a default user ID. Instead, the address of the ODBM address space is used for any security checks that later occur in IMS. See 3.5, “Security considerations” on page 73 for further detail about security methods used with the Open Database capability.

  - (RRS=Y): Two-phase commit is enabled by Resource Recovery Services (RRS).

- TCPIP=:
  - (HOSTNAME=TCPIP): The jobname of our TCP/IP host z/OS address space is TCPIP.
- ODACCESS=:
  - (ODBMAUTOCONN=Y): When our IMS Connect initializes, it automatically registers with all current and future ODBM instances. We can later disable this function by issuing the IMS Connect command SETOAUTO.
  - IMSPLEX=(MEMBER=IMSBBHWS,TMEMBER=PLEXB): The name of our IMS Connect instance within the IMSplex is IMSBBHWS. The IMSplex name is PLEXB and must be the same as the IMSPLEX name that is specified in the SCI initialization member, which you can see in Example 3-4 on page 47. If you are using OTMA for IMS Transaction Manager access and specified the name of the IMS data store using the ID parameter on the DATASTORE statement, keep in mind that it must be different than the value you define for the TMEMBER on this IMSPLEX statement.
  - (DRDAPORT=(ID=5555,PORTTMOT=6000)): The port number that IMS Connect will receive incoming requests for IMS data on is 5555 and must be unique among all other port specifications. Our IMS Connect instance waits 6000 hundredths of seconds for the next input message (after having already received the initial message) from a client application that is connected on a DRDA port. After this time elaspses, it disconnects the client, which is useful because it prevents IMS Connect from unnecessarily waiting for requests when a client is looping or hung. Not shown in our example is the KEEPAV parameter, which defines the frequency interval at which a packet is sent by the z/OS TCP/IP layer to port 5555, thus maintaining a connection when it is otherwise idle. Because we did not specify this parameter on the DRDAPORT statement, it is set to 0, meaning that we use the KeepAlive value defined in the z/OS TCP/IP stack.
  - (ODBMTMOT=6000): Our IMS Connect instance waits 6000 hundredths of seconds for each response message from ODBM and also for the initial message after a client application establishes a socket connection with it. When ODBM does not respond within 60 seconds, a message HWSJ2530W is sent to the client but the socket connection is retained. On the other hand, if the client application does not send data to IMS Connect after making the initial socket connection within 60 seconds, the connection is terminated.

Note:

- If you already have IMS Connect set up for use with other IMS capabilities, such as Open Transaction Manager Access (OTMA) or the IMS Control Center, you simply add the ODACCESS statement to your existing HWSCFGxx configuration member.
- If you are a DBCTL user and you are configuring IMS Connect for the first time, in the HWSCFGxx configuration member, the applicable statements that you must specify for using ODBM are: HWS, TCPIP, ODACCESS, and IMSPLEX statements.

For additional details about defining parameters in the *HWSCFGxx* member, refer to the manual *IMS Version 11 System Definition*, GC19-2444. If this is your first time setting up IMS Connect, we recommend reviewing the book *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity,* SG24-6794, because it has extensive details about IMS Connect functionality and additional set up recommendations. We are now ready to start our IMS Connect address space.

Starting the IMS Connect address space

Use the IMS Connect startup procedure to start the IMS Connect address space as a task, specifying the BPE configuration member and suffix of the IMS Connect configuration

member to be used. Refer to Example 3-12 on page 64 for a sample IMS Connect startup procedure.

Example 3-12 Sample startup procedure for the IMS Connect address space

```
//IMSBBHWS PROC RGN=OM,TME=1440,SOUT=S,
//             BPECFG=BPECFGIV,HWSCFG=HWSCFODB
//*
//* FUNCTION: START IMS CONNECT REGION.
//*
//HWSREGN  EXEC PGM=HWSHWS00,REGION=&RGN,TIME=&TME,
//              PARM='BPECFG=&BPECFG,HWSCFG=&HWSCFG'
//STEPLIB  DD   DSN=IMS11B.SDFSRESL,DISP=SHR
//*        DD   DSN=CEE.SCEERUN,UNIT=SYSALLDA,DISP=SHR
//*        DD   DSN=SYS1.CSSLIB,UNIT=SYSALLDA,DISP=SHR
//PROCLIB  DD   DSN=IMS11B.PROCLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=&SOUT
//SYSUDUMP DD   SYSOUT=&SOUT
//HWSRCORD DD   DSN=IMS11B.HWSRCDR,DISP=SHR
//*
```

In Example 3-12, we defined our BPE configuration member with BPECFG=BPECFGIV. This member is shown in Example 3-3 on page 46 and contains the tracing definitions that are used for this IMS Connect address space instance. We also specified HWSCFG=HWSCFODB, which is the IMS Connect configuration member that is shown in Example 3-11 on page 62 and is read when we run this procedure.

Tip: Make sure that your IMS Connect load library is APF-authorized and allow your IMS Connect to run in authorized supervisor state (key 7) by updating the program properties table (PPT) in SYS1.PARMLIB.

IMS Connect user exits

You can use several user exits with IMS Connect. With IMS Open Database, you can use the HWSROUT0 user exit to route an IMS data request to a specific IMS. Either the calling DRDA application or the IMS Universal Driver can specify this IMS, overriding the IMS alias that would have otherwise been selected. You can also use the exit routine to route requests to a specific ODBM instance during processing. For detail about this and other IMS Connect user exits, refer to *IMS Version 11 Exit Routines*, SC19-2437.

Tip: As a final step in setting up IMS Connect, install the default user exits into the IMS Connect resident library.

IMS Connect considerations for IMSplex

When defining an IMS Connect instance in an IMSplex environment, there are a few things to keep in mind:

- You are not limited to defining an IMS Connect instance to a single IMSplex. You can include multiple IMSPLEX statements within the HWSCFGxx configuration member to define a single IMS Connect instance to more than one IMSplex.
- An IMS Connect instance can register with an IMSplex with only one name. As previously stated, you can define an IMS Connect to multiple IMSplexes using the same name, but ensure that each statement designates a different IMSplex.

- Defining an SCI instance within the IMSplex enables communication between IMS Connect and the ODBM address space either within the same LPAR or cross-LPAR. When they reside on the same LPAR, SCI uses the z/OS Program Call (PC) function to enable communication, but on separate LPARs SCI uses the cross-system coupling facility (XCF).

3.3.2 Modifying existing IMS Connect definitions for IMS Open DB support

You might already be using IMS Connect for TCP/IP access to IMS transactions and commands. In this case, you already have IMS Connect installed. To use the IMS Open Database capability, just modify your IMS Connect configuration member HWSCFGxx to include the ODACCESS statement, as described in “HWSCFxxx” on page 62, and shown in Example 3-11 on page 62. After you update the configuration member, start IMS Connect as you usually do using JCL, similar to Example 3-12 on page 64.

3.4 Using IMS applications to help set up CSL and IMS Connect

The IMS product comes with two applications that can assist you in defining the PROCLIB members that are associated with the different IMS Open Database components that we just covered. In this section, we discuss the Installation Verification Program and the Syntax Checker applications.

3.4.1 Installation Verification Program

IMS provides a program that you can use to install a sample IMS system and try out different IMS functions. This program is called the IMS Installation Verification Program (IVP), and you can use it as a reference when setting up the required PROCLIB members for the CSL and IMS Connect address spaces. In it, there is a sample JCL that starts each of these address spaces.

Note: While the IVP starts the CSL and IMS Connect address spaces as jobs, you can also start them as started tasks.

The IVP also includes a sample application that exercises the IMS Open Database capability, which you can test if you want to use it as more than just a reference for defining IMS Open Database-related PROCLIB members and procedures.

Invoke the IVP using option 6 from the IMS Application menu shown in Figure 3-9 on page 66.

```
Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSAPPL                      IMS Application Menu
Command ===>

Select an application and press Enter.


         1   Single Point of Control (SPOC)
         2   Manage resources
         3   Knowledge-Based Log Analysis (KBLA)
         4   HALDB Partition Definition Utility (PDU)
         5   Syntax Checker for IMS parameters (SC)
         6   Installation Verification Program (IVP)
         7   IVP Export Utility (IVPEX)
         8   IPCS with IMS Dump Formatter (IPCS)
         9   Abend Search and Notification (ASN)


 To exit the application, press F3.

 F1=Help   F12=Cancel
```

Figure 3-9 The IMS Application Menu

Follow the prompts until you reach Execution mode, which lists a series of jobs and tasks that start the sample IMS system and test different functions. In our case, we selected a Database and Transaction Management (DB/DC) environment and the IMS Connect and Open Database Sample sub-options, as shown in Figure 3-10 on page 67.

```
Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
IVP                 Sub-Option Selection - DBT                 IMS 11.1
Command ===>


Select the desired Sub-Options and press ENTER

   IRLM - Use IRLM in IVP Applications
   FP   - Use Fast Path in IVP Applications
   ETO  - Use Extended Terminal Option
   CQS  - Add CQS to CSL Applications
   RACF - Use RACF Security
   JAVA - Use JAVA Applications
   PRA  - Use Parallel RECON Access
/  ICON - Use IMS Connect
/  OPDB - Use Open Database Sample

  Note: Your Sub-Option selection affects the user variables, jobs, and tasks
  that will be presented.  If you later change your selection, you must redo
  the IVP Table Merge, Variable Gathering, File Tailoring, and Execution
  processes. RACF is required when Java sub-option is selected.

 F1=Help      F3=Exit      F7=Backward  F8=Forward  F10=Actions
```

Figure 3-10 Sub-options of IMS Connect and the Open Database sample in the IVP

Sample PROCLIB member definitions

Using the IVP, you can view sample PROCLIB member definitions for all of the components that are associated with the IMS Open Database capability. Scroll down to the `IV3E` series of jobs/tasks within the Execution panel, and look for the IV3E302J and IV3E303J jobs, as shown in Figure 3-11 on page 68.

```
 Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                       Execution (LST Mode) - DBT          Row 32 to 47 of 215
Command ===>                                                  Scroll ===> PAGE

Action Codes: Brm Doc Edm eNt eXe Ftl spR
      JOB/Task    Step  Title.........................................
      IV3E001T    E0    NOTE - Introduction - Build IVP Appl / System
      IV3E101J    E1    JOB  - Allocate Data Sets
      IV3E201J    E2    JOB  - DBDGENs
      IV3E202J    E2    JOB  - PSBGENs
      IV3E203J    E2    JOB  - ACBGEN
      IV3E204J    E2    JOB  - MFS Language Utility
      IV3E206J    E2    JOB  - Assembly/Bind Applications
      IV3E207J    E2    JOB  - Assembly/Bind Install. Default Block
      IV3E301J    E3    JOB  - Create Dynamic Allocation Members
*     IV3E302J    E3    JOB  - Add Control Statements to IMS.PROCLIB
*     IV3E303J    E3    JOB  - Add SCI/OM/RM Members to IMS.PROCLIB
      IV3E305J    E3    JOB  - Define EXEC PARM Defaults
      IV3E306T    E3    TASK - Syntax Checker Sample
      IV3E307T    E3    TASK - Define z/OS Policies
      IV3E308J    E3    JOB  - Define DRA Start-up Table
      IV3E309J    E3    JOB  - Verify TCO Scripts
 F1=Help      F3=Exit      F7=Backward  F8=Forward  F10=Actions
```

Figure 3-11 The IV3E302J and IV3E303J jobs show examples adding required PROCLIB members

BPE, ODBM, and IMS Connect samples

If you browse the IV3E302J job, you can see several examples of the IVP-defining members to IMS PROCLIB:

- HWSCFODB specifies IMS Connect configuration statements.
- CSLDI001 and CSLDC001 define ODBM initialization and configuration statements, respectively.
- BPEODBM is the sample BPE configuration member, which is specified later in the IVP's sample IMS Open Database application when starting the ODBM address space.

OM, SCI, and DFSCGxxx samples

Browse the IV3E303J job to see sample PROCLIB definitions for the SCI and OM initialization members: CSLOI000 and CSLSI000, respectively. Within this job, there are two examples that show how to define a DFSCGxxx PROCLIB member, which is important in designating the IMSplex name that all CSL components are a part of.

Attention: As of IMS V10, you can specify your CSL definitions in the CSL section of the DFSDFxxx PROCLIB member. If the CSL definitions are present in both DFSDFxxx and DFSCGxxx members, those that are contained in DFSCGxxx take precedence.

We saw examples of how the Open Database components' initialization and configuration members are defined to IMS PROCLIB, now we review how to find sample jobs that start the associated address spaces within the IVP application.

Sample startup JCL

You can find the jobs that start SCI, OM, ODBM, and IMS Connect in the `IV3T` series of IVP jobs shown in Figure 3-12.

```
 Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                       Execution (LST Mode) - DBT        Row 194 to 209 of 215
Command ===>                                                  Scroll ===> PAGE

Action Codes: Brm Doc Edm eNt eXe Ftl spR
      JOB/Task    Step  Title.....................................
      IV3T101T    T1    NOTE - Introduction - Open Database sample
      IV3T101J    T1    JOB - Allocate Data Sets
      IV3T102J    T1    JOB - Initialize RECON
      IV3T103J    T1    JOB - Register Data Bases
      IV3T104J    T1    JOB - Data Base Initial Load
      IV3T105J    T1    JOB - Batch Image Copy
      IV3T201T    T2    TASK - Start TCP/IP And RRS
*     IV3T201J    T2    JOB  - Start SCI
*     IV3T202J    T2    JOB  - Start OM
      IV3T203J    T2    JOB  - Start RM
      IV3T204J    T2    JOB  - Start IRLM
      IV3T206J    T2    JOB  - Start DB/DC
      IV3T208T    T2    TASK - Cold Start IMS DB/DC
*     IV3T210J    T2    JOB - Start ODBM
*     IV3T211J    T2    JOB - Start IMS Connect
      IV3T220J    T2    JOB - Create a Unix Script to run the application
      IV3T230J    T2    JOB - Run the sample and copy the output to joblog
F1=Help      F3=Exit      F7=Backward  F8=Forward  F10=Actions
```

Figure 3-12 The IV3T series of the IVP contains jobs that start IMS Open Database components

For more information about how to use the IVP, refer to the manual *IMS Version 11 Installation*, GC19-2438.

3.4.2 IMS Syntax Checker

You can also use the IMS Syntax Checker to assist you in defining, verifying, and validating the parameters that are contained in your PROCLIB members. In this section, we use the IMS Connect configuration member HWSCFODB as an example and show you how to use Syntax Checker to validate the settings that are contained in it.

1. Invoke the IMS Application Menu, and select option **5** for the IMS Syntax Checker shown in Figure 3-9 on page 66.
2. On the next panel, enter the name of the data set that contains the member definitions that you want to validate with the Syntax Checker, as shown in Figure 3-13 on page 70. Our members are contained within the IMS11B.PROCLIB data set, so we specified it here with single quotes.

```
 File  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT                IMS Parameter Syntax Checker     Enter required field
Command ===>

Enter the name of the IMS proclib dataset and press enter.


 ISPF Library:
    Project  . .
    Group  . . .
    Type . . . .
    Member . . .           (Blank for member list)



 Other Partitioned Data Set:
    Data Set Name  . . 'IMS11B.PROCLIB'
    Volume Serial  . .           (If not cataloged)


 F1=Help    F3=Exit   F12=Cancel
```

Figure 3-13 Specifying the data set name containing our member definitions to be validated

3. The next panel displays a member list, select a member to review, as shown in Figure 3-14. In our example, we want to check the syntax of our IMS Connect configuration member, HWSCFODB.

```
 Menu  Functions  Utilities  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT ST  IMS11B.PROCLIB                                 Row 00046 of 00093
Command ===>                                                  Scroll ===> PAGE
   Name     Prompt          Size    Created           Changed            ID
. DFSVSMDC                   14   2010/02/25   2010/02/25 20:51:52    IMSR2
. DFSVSM00                    8   2010/02/25   2010/02/25 20:52:29    IMSR2
. DFSWT000
. DLIBATCH
. FDR
. FMTCPY
. FPUTIL
. HWSCFGIV
S HWSCFODB                    4   2010/02/22   2010/03/03 18:02:01    IMSR3
. HWSCFODX                    6   2010/02/22   2010/02/22 15:27:55    IMSR1
. ICJCL
. ICRCVJCL
. IMS
. IMSBATCH
. IMSBBDL1
 F1=Help    F2=Split   F3=Exit    F5=Rfind   F7=Up      F8=Down    F9=Swap
F10=Left   F11=Right  F12=Cancel
```

Figure 3-14 Selecting the IMS Connect configuration member for review within Syntax Checker

4. The next panel prompts you to indicate whether this member is a BPE exit list member or an IMS Connect configuration member. Select the latter, as shown in Figure 3-15.

```
 File  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT                IMS Parameter Syntax Checker
Command ===>

Enter the following information and press enter.

Select member type . . . . . 2  1. BPE Exit List Member
                                2. IMS Connect Configuration Member









 F1=Help    F3=Exit   F12=Cancel
```

Figure 3-15 The Syntax Checker requesting input regarding the member type

5. Select IMS 11.1 from the next menu, as shown in Figure 3-16.

```
 File  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT                IMS Parameter Syntax Checker
Command ===>

Enter the following information and press enter.

IMS Release  . . . . . 1  1. IMS 11.1
                          2. IMS 10.1
                          3. IMS 9.1









 F1=Help    F3=Exit   F12=Cancel
```

Figure 3-16 Syntax Checker prompting for IMS version level

6. From the list of values that the Syntax Checker displays, define within the member with a brief explanation of each parameter's function. We scrolled down to the ODACCESS section of the HWSCFODB member in our sample shown in Figure 3-17. You can alter the display by choosing to collapse or expand certain sections.

```
 File  Edit  View  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT           IMS 11.1 Parameters for ANY
Command ===>

Press enter (without other input) to check for errors.

Data Set Name  . . : IMS11B.PROCLIB(HWSCFODB)
IMS Release  . . . : 11.1

  Sel Codes: C = Comment D = Delete I = Insert
             P = Process + = Expand - = Contract / = Select

 Sel Keyword                      Description             More: -+
  _  ODACCESS (                   ODACCESS statement
  _   DRDAPORT = (                Define ports
  _    ID = 5555                  Port number
  _    KEEPAV =                   KeepAlive interval seconds
  _    PORTTMOT = 6000       )    Timeout value in hundredths of a second
  _   DRDAPORT = (                Define ports
 F1=Help      F3=Exit      F6=Default   F7=Backward  F8=Forward  F10=Edit
F12=Cancel
```

Figure 3-17 Syntax Checker view of our IMS Connect configuration member, HWSCFODB

One very useful feature within Syntax Checker is the F1=Help function. If you place your cursor on a field and press F1, it displays detailed information about the parameter meaning, and what values are valid. Refer to Figure 3-18 on page 73 for an example of the help panel for the PORTTMOT parameter.

```
 File  Help
sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
DFSSCSRT           HELP FOR PORTTMOT
Command ===>

                                                                      More  +
Keyword: PORTTMOT

      PORTTMOT=
         Defines the amount of time that IMS Connect waits for the next
         input message from a client application that is connected on a
         DRDA port before IMS Connect disconnects the client.

         The timeout interval is specified as a decimal integer in
         hundredths of a second. Valid values for PORTTMOT are from 0 to
         2 147 483 647 (X'7FFFFFFF'). The default is 18 000 (3 minutes).
         A 0 disables the timeout function.



         Specifying a timeout value can avoid hang conditions when a
         client stops sending messages as expected due to, for example:
         *  A looping client application
 F1=Help      F3=Exit      F7=Backward  F8=Forward  F12=Cancel
```

Figure 3-18 Help panel for the PORTTMOT parameter of the ODACCESS statement

3.5 Security considerations

When a data request comes into IMS through ODBM from either a DRDA application or one of the IMS Universal Drivers, it must go through security checking before the data can be retrieved. This process is two-fold. First, use IMS Connect security to authenticate the user before the request reaches ODBM. This authentication confirms that users really are who they claim they are. Second, determine whether the user is authorized to access the IMS resource using either allocate PSB (APSB) security or Resource Access Security (RAS). ODBM can retrieve the requested data from IMS only if these security checks are passed.

We now explore how to use IMS Connect to authenticate a user ID. Then we discuss how to authorize this user to allocate a PSB or access an IMS resource using the APSB and RAS facets of IMS security.

Authenticating a user ID using IMS Connect security

IMS Connect security gives you two options to authenticate users who are requesting IMS data: using the Security Authorization Facility (SAF) interface or the IMS Connect DB Security user exit routine, HWSAUTH0.

Using the Security Authorization Facility

IMS Connect can call a SAF interface, such as RACF, to authenticate a user. To activate this option, define RACF=Y in the HWSCFGxx configuration member (this setting can later be changed using the IMS Connect SETRACF command). When an application sends in a request for IMS data, it will include a user ID (if it is not being provided by an exit) and either a password or a RACF PassTicket.

RACF PassTicket: A RACF PassTicket is a one-time-only password that is generated by the calling application or a user exit and is an alternative to the RACF password. This is considered to be more secure than using the RACF password because it removes the need to send the password across the network in clear text.

IMS Connect then performs authentication with a call to RACF. When the request reaches ODBM, the user must already be authenticated and only then needs to be authorized to access the APSB or the particular IMS resource.

IMS Connect DB Security user exit (HWSAUTH0)

Instead of SAF, you can use the IMS Connect DB Security user exit routine (HWSAUTH0) to perform the necessary user ID security checking when a message is received from an application program. This refreshable exit is shipped with IMS Connect and link-edited into the IMS Connect RESLIB, and will always be called even if RACF=Y is specified and RACF is called afterwards.

It can authenticate a user ID that was passed in by an application, and can even designate a separate user ID to replace the original before it is sent to ODBM. The exit can also provide a RACF group ID to be authenticated further by IMS Connect.

HWSAUTH0 is a standard BPE type-1 exit routine, and you can manage it using the BPE DISPLAY USEREXIT and REFRESH USEREXIT commands. To use the exit, you must first create or modify your current BPE exit list PROCLIB member using a name of your choice. We used HWSICNX0. Also in the member, define HWSAUTH0 as an exit similar to what we show in Example 3-13. The example contains all values exactly as you must code them, but you can specify a number of your choice for the ABLIM= parameter, which tells IMS how many times the exit can abend before it becomes disabled.

Example 3-13 Defining the HWSAUTH0 user exit within the BPE exit list PROCLIB member

```
EXITDEF(TYPE=ODBMAUTH,EXITS=(HWSAUTH0),ABLIM=8,COMP=HWS)
```

Finally, add an EXITMBR statement to your BPE configuration member, specifying the name of your new exit, similar to our sample shown in Example 3-14. We designated that our new exit, HWSICNX0, be used.

Example 3-14 Adding a user exit to the BPE configuration member with the EXITMBR statement

```
EXITMBR=(HWSICNX0,HWS)      /* IMS CONNECT EXITS */
```

Authorizing a user ID to access a resource using IMS security

After a user ID is authenticated by IMS Connect, the request for IMS data flows to ODBM, which in turn passes the request to the appropriate IMS system. At this point, we must determine whether the user ID is authorized to allocate the PSB or access another IMS resource. The path we take here depends on what value we specified for our ODBASE= execution parameter in our DFSPBxxx PROCLIB member. If we specified ODBASE=Y and our ODBM is running with RRS, we use APSB security; otherwise, if we specified ODBASE=N (the default) or our ODBM is not running with RRS, we use RAS security.

Attention: If you choose to track statistics for a RACF user ID, keep in mind that performance can be affected.

Let us now look at how each of these security types function.

Allocate PSB (APSB) security

To protect APSBs from unauthorized access, protect them in the AIMS RACF class and grant the user access to each protected resource. See Example 3-15 for a simple model of what these RACF definitions might look like.

Example 3-15 Sample RACF definitions for authorizing a user to access a protected APSB

```
ADDUSER APPLUID1
RDEFINE AIMS APSBX UACC(NONE)
PERMIT APSBX CLASS(AIMS) ID(APPLUID1) ACCESS (READ)
```

In our example, the user ID of our calling application is APPLUID1. We protect an APSB named APSBX in the AIMS RACF class with an RDEFINE statement. Then we grant our APPLUID1 user access to this APSB with a PERMIT statement. Before the PSB can be dynamically allocated, the application's user ID is first authorized against the APSB.

Restriction: You can only use APSB security if ODBM is running with Resource Recovery Services (RRS), which is the default setting.

Resource access security (RAS)

As we previously mentioned, RAS is used if you specified ODBASE=N in either your IMS startup procedure or your DFSPBxxx PROCLIB member. It is also used if your ODBM is running without RRS. RAS determines whether the user that passed to ODBM is authorized to access a requested PSB. You can specify the type of security that RAS uses with the ISIS parameter, which is also defined in either the IMS startup procedure or the DFSPBxxx member. Here are the possible values that you can define for the ISIS parameter:

- ISIS=N disables RAS security
- ISIS=R uses RACF for RAS security
- ISIS=C uses the DFSRAS00 user exit for RAS security
- ISIS=A uses both RACF and the DFSRAS00 user exit for RAS security

Note: In previous IMS releases, you specified ISIS=0, ISIS=1, or ISIS=2, which were related to SMU AGN security. After IMS V9, SMU security is no longer supported, and if you specify any of these no longer supported values, they are internally translated to ISIS=N, and RAS is disabled.

You can also activate RAS with the SECURITY macro's TYPE parameter. Here are the possible values:

- NORAS disables RAS security
- RASRACF uses RACF for RAS security
- RASEXIT uses the DFSRAS00 user exit for RAS security
- RAS uses both RACF and the DFSRAS00 user exit for RAS security

Important: The ISIS parameter always overrides any RAS-related specification on the SECURITY macro. To use the SECURITY macro's designation, you must omit the ISIS parameter entirely.

For more information about the SECURITY macro, refer to *IMS Version 11 System Definition*, GC19-2444.

When you use RAS with RACF, the user ID is authorized to access a PSB protected in the RACF class IIMS (or the JIMS RACF grouping class for PSBs). The user ID that is passed to ODBM can vary depending on its source, which can be from any of these items:

- IMS Connect: The user ID is passed with the request and is typically that of the application.
- An ODBA application, such as WebSphere Application Server or DB2 stored procedure: The user ID is that of the user.
- A batch ODBA application: The user ID is the batch job's user ID.
- If no user ID was passed on the ODBM request: The user ID is the ODBM address space's user ID, specified on the USER= parameter of the ODBM address space's startup JCL.

If you specify that both RACF and the exit are used, the exit is always called after RACF is called. For more information about how to use the DFSRAS00 user exit routine, refer to the manual entitled *IMS Version 11 Exit Routines*, SC19-2437.

4

Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility

In this chapter, we explain the functionalities that the IMS Enterprise Suite offers for Open Database support. The IMS Enterprise Suite DLIModel Utility is required to generate the IMS database metadata file.

This chapter is intended to be read by the person responsible for generating the IMS metadata files. Depending on how the tasks are assigned in your organization, this can be the IMS system programmer, the IMS database administrator, or the application developer who has knowledge of IMS and has access to the necessary resources.

Several SOA technologies (both existing and as improved by IMS 11) are now packaged in the *IMS Enterprise Suite* product. IMS Enterprise Suite V1.1 (program numbers 5665-T60 and 5665-T61) contains the following items:

- IMS SOAP Gateway
- IMS Connect APIs for Java and C
- JMS API
- DLIModel Utility

In this chapter, we concentrate on the new IMS Enterprise Suite DLIModel utility's functions, which we cover in the following order:

- Introduction to the IMS Enterprise Suite
- Overview of the IMS Enterprise Suite DLIModel utility
- Downloading and installing
- Setting up for sample scenarios included in this book
- Using the IMS Enterprise Suite DLIModel utility
- Additional considerations for the IMS Enterprise Suite DLIModel Utility

4.1 Introduction to the IMS Enterprise Suite

The IMS Enterprise Suite consists of components that are designed to support open integration technologies to enable new application development and to extend the access to IMS transactions and data. Here, we focus on a single component of the IMS Enterprise Suite: The IMS Enterprise Suite DLIModel utility plug-in (also referred to as the DLIModel utility). The plug-in is based on Eclipse and can therefore be integrated into an existing Eclipse development environment.

After installed, it can take existing IMS PSB and DBD source and input them into the DLIModel utility to create the metadata. You can then use this metadata to write Java applications in Eclipse, IBM Rational Application Developer for WebSphere, or IBM Rational Developer for System z.

In this chapter, we show you how to download, install, and use the DLIModel utility for this purpose. First, we provide a more detailed overview of the utility, including its requirements, restrictions, and a brief history of its evolution.

4.2 Overview of the IMS Enterprise Suite DLIModel utility

IMS data is traditionally described in mainframe-based IMS source files, such as program specification blocks (PSBs) and database descriptions (DBDs). To use this data in a modern Java application, it must first be transformed into a format that the application can understand. As of IMS V11, IMS does not have a catalog that contains these transformed definitions. Rather, the DLIModel utility is the mechanism that performs this transformation. It takes IMS source files, such as PSB libraries, DBD libraries, and COBOL and PL/I copybooks as input and generates metadata, which describes the IMS data. More specifically, it generates a Java class called the IMS Java metadata class, which is a subclass of the com.ibm.ims.db.DLIDatabaseView class. This class can then be used within a Java application to access the IMS data.

Not only can the DLIModel utility generate metadata for use with IMS Open Database, but it can also generate elements that you can use with other functions, such as exposing IMS database operations as Web Services, for example, the utility can:

- Generate a graphical UML model that illustrates the IMS database structure in an interactive tree model
- Generate annotated XML schemas of IMS databases, which are used to retrieve XML data from or store XML data in IMS databases
- Incorporate additional field information from COBOL or PL/I copybooks
- Incorporate additional PCB, segment, and field information, or override existing information through a graphical view of the PCB
- Generate a DLIModel database report, which assists Java application programmers in developing applications based on existing IMS database structures
- Generate an optional DLIModel trace log
- Provide a configuration editor as a launching point for generating IMS database Web Services artifacts
- Generate XMI descriptions of the PSB and its databases

Figure 4-1 on page 79 is a representation of the input that the DLIModel utility accepts and what it generates as output.

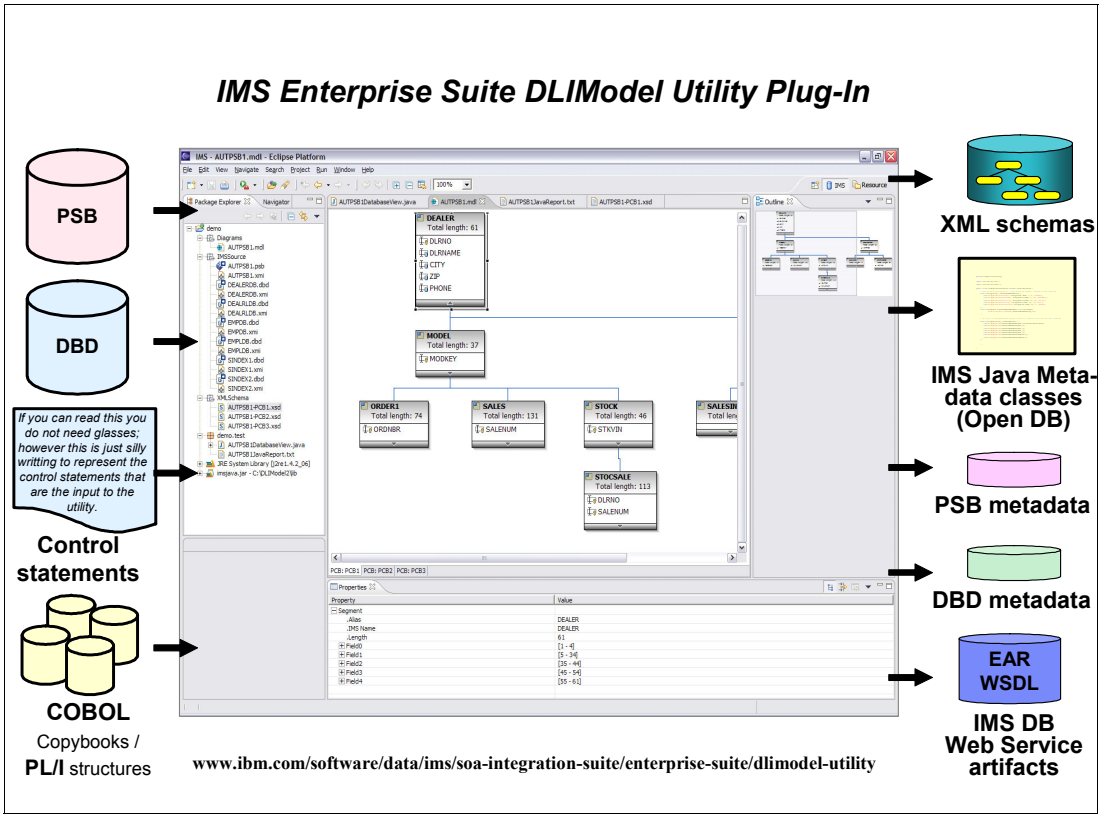


Figure 4-1 Input and output associated with the IMS Enterprise Suite DLIModel utility

For more information about the utility's generation capabilities:

1. Access the online IMS Information Center at:

   `http://www.ibm.com/ims`

2. At the Information Center, enter the search string `DLIModel utility`.
3. To locate the Information Center from the IMS homepage, click the Library link on the left, as shown in Figure 4-2 on page 80.

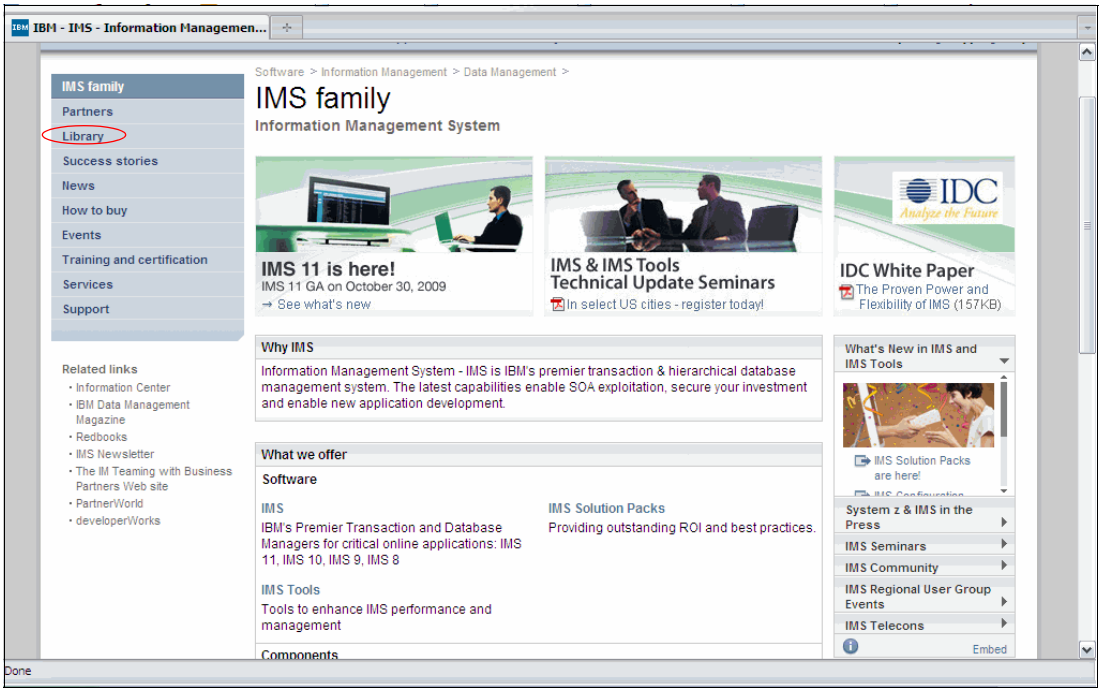


Figure 4-2 IMS family main panel

4. Click the **Information centers** link shown on the next page. Now that we described a bit about the DLIModel utility's capabilities, we now look at its requirements.

4.2.1 Requirements

To use the DLIModel utility, certain software requirements must be met. This includes the scenario where you are migrating from an older version of the utility to the current one. You must be using:

- Eclipse Version 3.4.1
- Graphical Editor Framework (GEF) 3.4.1
- Eclipse Modeling Framework (EMF) 2.4.2

There are also other general requirements that must be met before you can use the DLIModel utility:

- Eliminate all errors from your PSB and DBD source code, and from your COBOL or PL/I copybooks before running the utility because it does not validate the input data.
- Include a name for all PCBs in each PSB definition using statement labels or the PCBNAME parameter.
- Ensure that all PSBs and DBDs, either directly or indirectly related to resources that are included in the IMS source files, are accessible. Include DBDs that are indirectly referenced by a PSB in the case of secondary indexing on a main database and DBDs that are externally referenced by other DBDs because of a logical relationship.
- Specify PROCOPT=P in your PCB statements or in the SENSEG statement for the root segment during PCB generation, if your application includes JDBC calls that span more than one segment in a hierarchical path.
- When your application is using IMS Java hierarchical database interfaces (SSA database access) to retrieve IMS data, ensure that you choose appropriate PSB processing options because you are controlling path calls.

- Keep track of the length field when you have variable length segments for INSERT or UPDATE statements.
- Ensure that your COBOL and PL/I copybook files that supply additional information about field layouts describe physical segments. The files cannot describe logical database segment layouts.
- After you check and change your IMS definitions, as previously mentioned, copy the source files to your distributed environment.

Now that we are familiar with the requirements for using the DLIModel utility, we now review restrictions that you must adhere to when using the utility.

4.2.2 Restrictions

When using the DLIModel utility to transform IMS data, there are a few items to keep in mind. We previously mentioned that the utility can process PSB lsource, DBD lsource, and COBOL and PL/I copybooks.

You can only import the copybooks if you installed the utility plug-in into IBM Rational Application Developer for WebSphere Software or IBM Rational Developer for System z Version 7.5 or later.

The DBD source can process all database organizations except MSDB, HSAM, and SHSAM databases. It can process all types and implementations of logical relationships and secondary indexes except for shared secondary indexes. It is not able to process the PROCOPT=K option in a PSB SENSEG statement, which makes an application sensitive only to the segment key of a segment.

To access IMS data on the mainframe, you must first acquire the respective source files (DBDs and PSBs) from the z/OS system using a means to copy them to the distributed platform, for instance, using FTP or the Remote System Explorer feature of RDz. The DLIModel utility cannot change the source files of DBDs and PSBs.

The DLIModel utility does not use DLITypeInfoList classes in its generated classes. If you want to define repeating groups of fields in segments other than by explicitly defining each group of fields separately, you must create the classes manually or modify the classes that that the DLIModel utility generates.

Field type will vary depending on how it was created. When it is defined by a DBD, it is a CHAR field type, but when it is imported from a COBOL or PL/I copybook, it is defined by the COBOL or PL/I copybook. You can change the field type by modifying it manually.

4.2.3 History

Prior to IMS 11, the DLIModel utility was shipped with IMS and ran from UNIX System Services or from the z/OS BPXBATCH utility. It required knowledge of z/OS development and also required writing control statements. However, this version of the utility is no longer supported (after IMS 10), so your option for obtaining the utility is through web download or by ordering the IMS Enterprise Suite through the standard ordering process.

The web download version of the DLIModel utility was also available prior to IMS 11 as a plug-in, but it was packaged differently in that it was not yet part of the IMS Enterprise Suite. It was simply called the IMS DLIModel utility plug-in. This version of the utility is still available for download, but it does not include enhancements made to the newly packaged version of the utility, called the IMS Enterprise Suite DLIModel utility. These latest enhancements include a

shell-sharing capability with Rational Application Developer Software for WebSphere or Rational Developer for System z. We therefore recommend that you use the latest version of the DLIModel utility, and install it using the IBM Installation Manager, so that you can to take advantage of the most current utility features.

4.3 Downloading and installing

To obtain the IMS Enterprise Suite DLIModel utility plug-in, you must use the IBM Installation Manager and configure its installation repository:

1. Begin by navigating to the IMS homepage located at:

   `http://www.ibm.com/ims`.

2. On this page, scroll down to find the IMS SOA Integration Suite link, as shown in Figure 4-3.

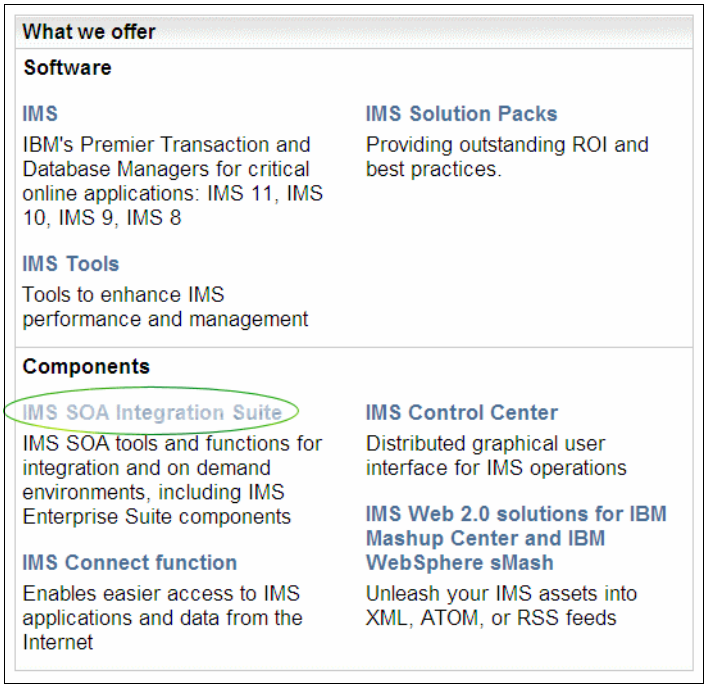


Figure 4-3 Starting point for downloading the IMS Enterprise Suite DLIModel utility plug-in

3. Follow the **IMS Enterprise Suite** link on the next two panels to navigate to the IMS Enterprise Suite download site. You are prompted to sign in with your IBM ID (or create one if you do not yet have an IBM ID) and agree to the license terms before proceeding. The next page presents a list of items that you can select for download, as shown in Figure 4-4 on page 83.
4. To download both the IBM Installation Manager and the IMS Enterprise Suite DLIModel utility components using Download Director, select the following items in the list:
   - Installation Manager for Windows
   - IMS Enterprise Suite DLIModel utility plug-in for Red Hat® Linux and Windows XP
   - Agreement to the license terms
5. At the bottom of the page, click **I Confirm**.

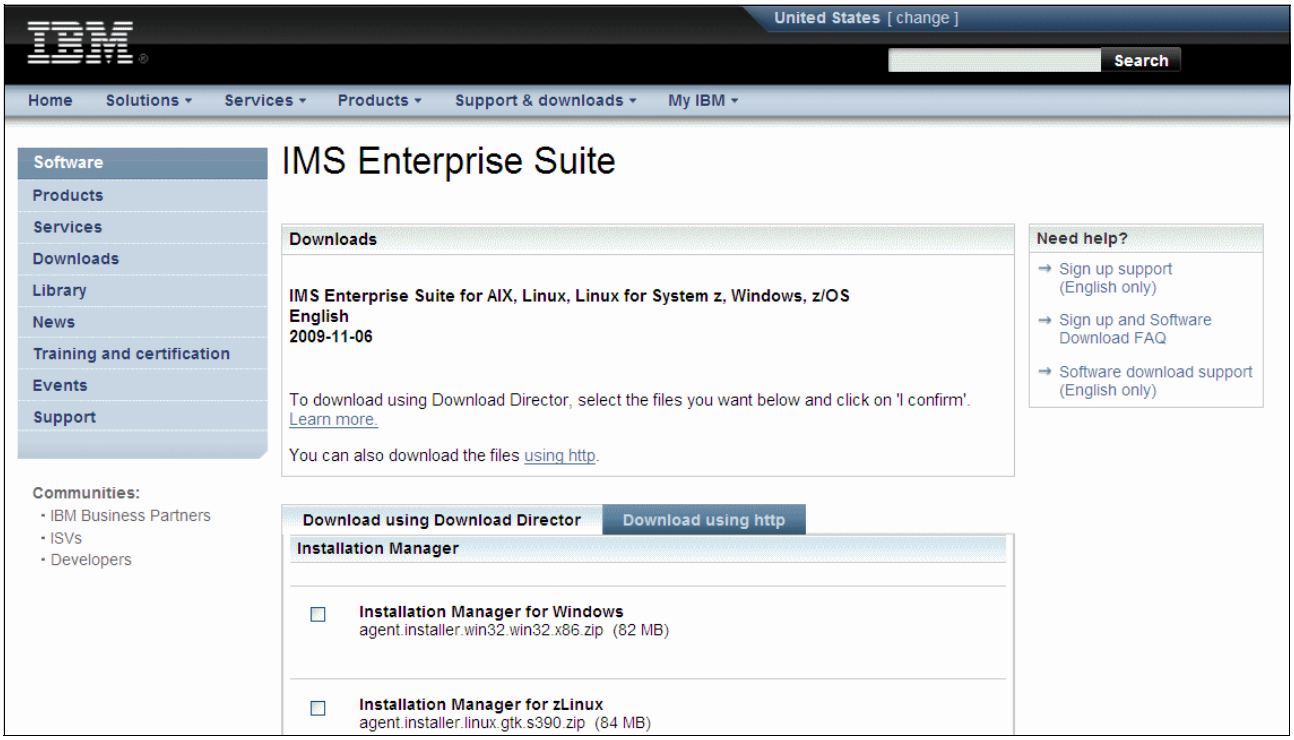


Figure 4-4 Download page for the IBM Installation Manager and IMS Enterprise Suite DLIModel utility plug-in

In Figure 4-4, there is one item in the list entitled Contains all IMS Enterprise Suite features. SMP/E installable" under the "IMS Enterprise Suite (SMP/E)" heading. The DLIModel utility is part of this particular item; however, if you select this, all of the other IMS Enterprise Suite components are downloaded too (and can be installed through SMP/E). We recommend that you select the individual DLIModel utility link as mentioned previously because the other IMS Enterprise Suite components do not play a role in the IMS Open Database capability.

After you click **I confirm**, two files are downloaded:

- agent.installer.win32.win32.x86.zip (IBM Installation Manager)
- dlimodel_distribute_repository.zip (IMS Enterprise Suite DLIModel Utility)

Next we review the installation procedures for each of these files.

4.3.1 Installing the IBM Installation Manager

To install the IBM Installation Manager:

1. Open the downloaded agent.installer.win32.win32.x86.zip file, and click **install.exe** to initiate the installation process, as highlighted in Figure 4-5 on page 84.

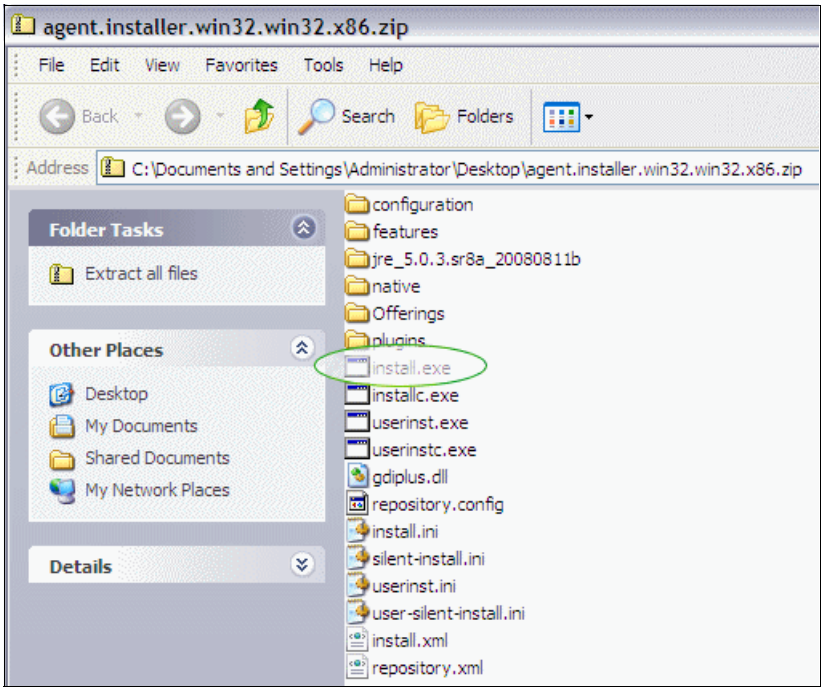


Figure 4-5 The install.exe file initiates the IBM Installation Manager installation process

2. Follow the prompts to complete the installation, and agree to the terms of the license agreement. After the IBM Installation Manager successfully installs, Figure 4-6 is displayed.

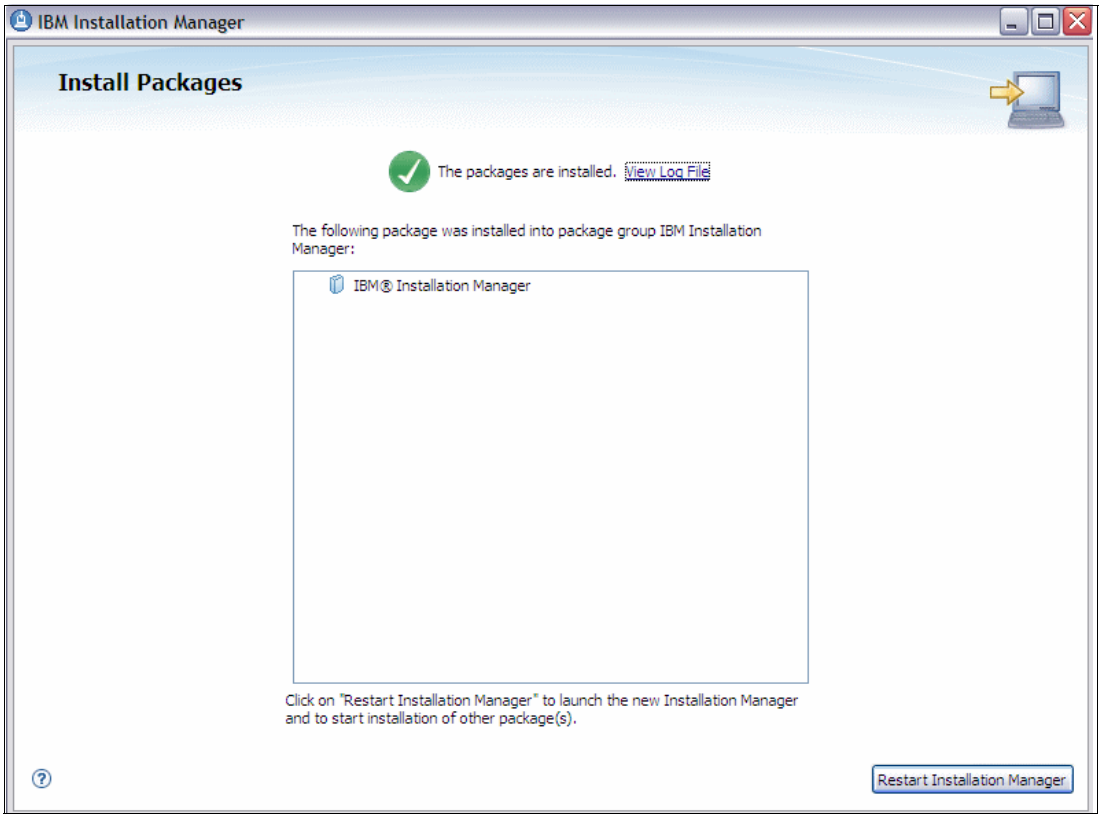


Figure 4-6 The confirmation panel after the IBM Installation Manager is installed

Attention: Any program installed by the IBM Installation Manager does not appear in the standard Windows Control Panel “Add/Remove Programs” list. You must use the IBM Installation Manager to remove any program that it installed.

4.3.2 Installing the IMS Enterprise Suite DLIModel utility

To install the IMS Enterprise Suite DLIModel utility:

1. Now that you installed the IBM Installation Manager, launch it to install the IMS Enterprise Suite DLIModel utility. Figure 4-7 shows the IBM Installation Manager’s main menu.

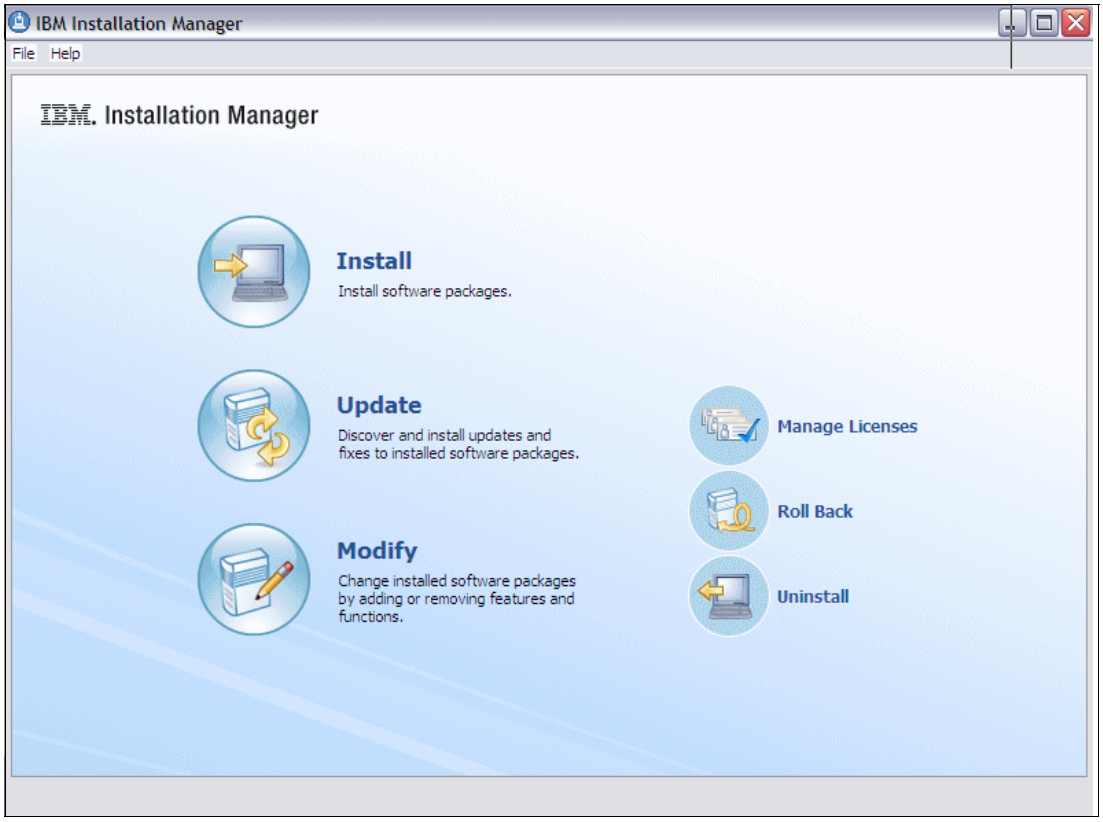


Figure 4-7 The main menu displayed when the IBM Installation Manager is launched

2. When you use the IBM Installation Manager to install software packages, you must connect to repositories that contain these software packages. To connect to the repository that is associated with the IMS Enterprise Suite DLIModel utility, from the File menu, select **Preferences**, click **Add Repository**, and select the **dlimodel_distribute_repository.zip** file that you just downloaded from the web page in the previous section to add it to the dialog, as shown in Figure 4-8 on page 86.

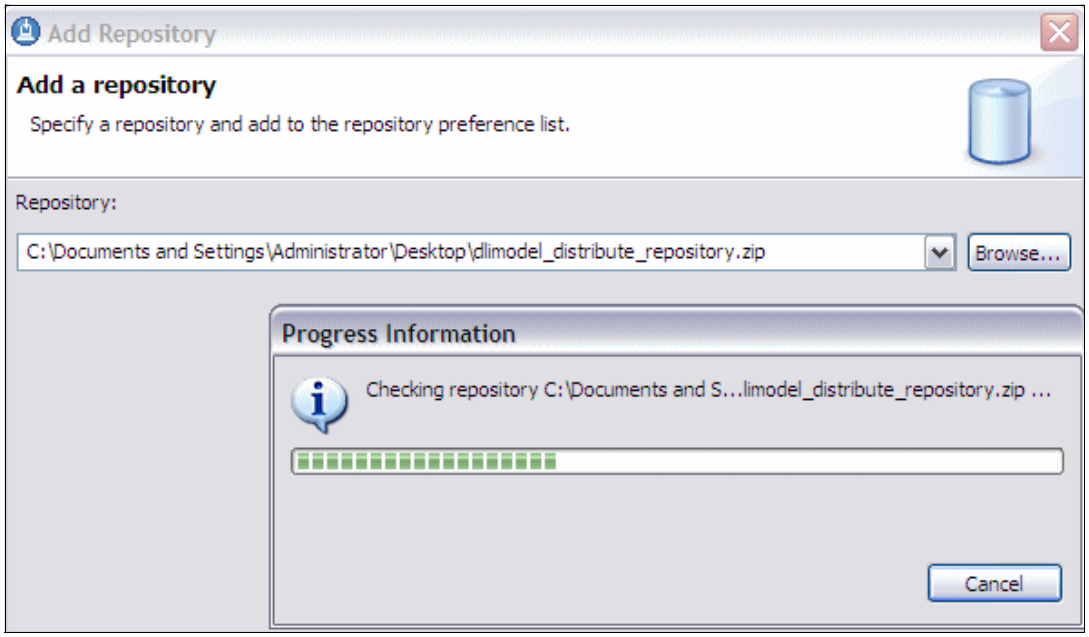


Figure 4-8 Adding the DLIModel utility repository to the IBM Installation Manager

3. Now that you added the repository, click **OK**. After a few seconds you are returned to the main menu of the IBM Installation Manager, where you can click **Install**. The next panel displayed contains the packages that are available to install. Select **IMS Enterprise Suite DLIModel Utility Plug-in**, as shown in Figure 4-9.

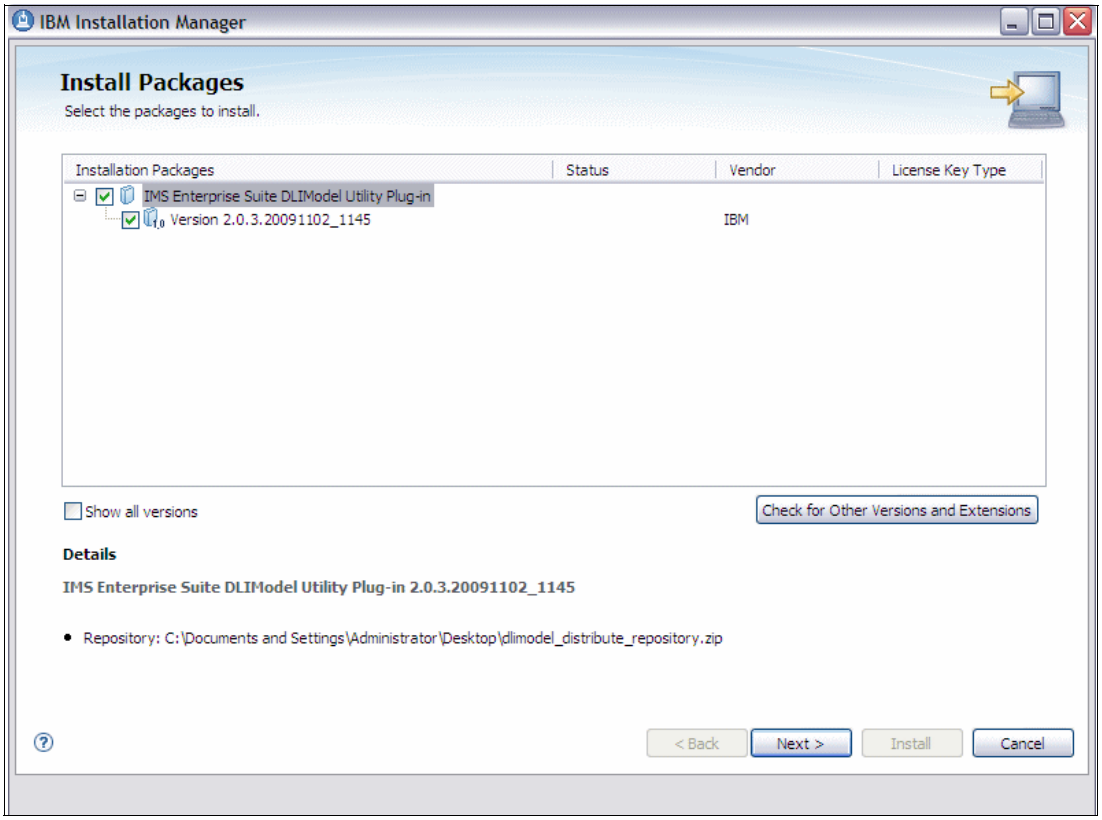


Figure 4-9 Selecting the IMS Enterprise Suite DLIModel Utility Plug-in package for installation

4. Click **Next**, accept the terms of the license agreement on the next page, and follow the rest of the dialog prompts to complete the installation process.

Installing maintenance for the DLIModel Utility

When adding the latest fix pack to the DLIModel Utility, the procedure is very similar to the installation:

1. You can download the fix pack from the same IBM web site where you can download the IMS Enterprise Suite DLIModel Utility. The fixable is a zip file that you can use as repository for the IBM Installation Manager, as shown in Figure 4-10.

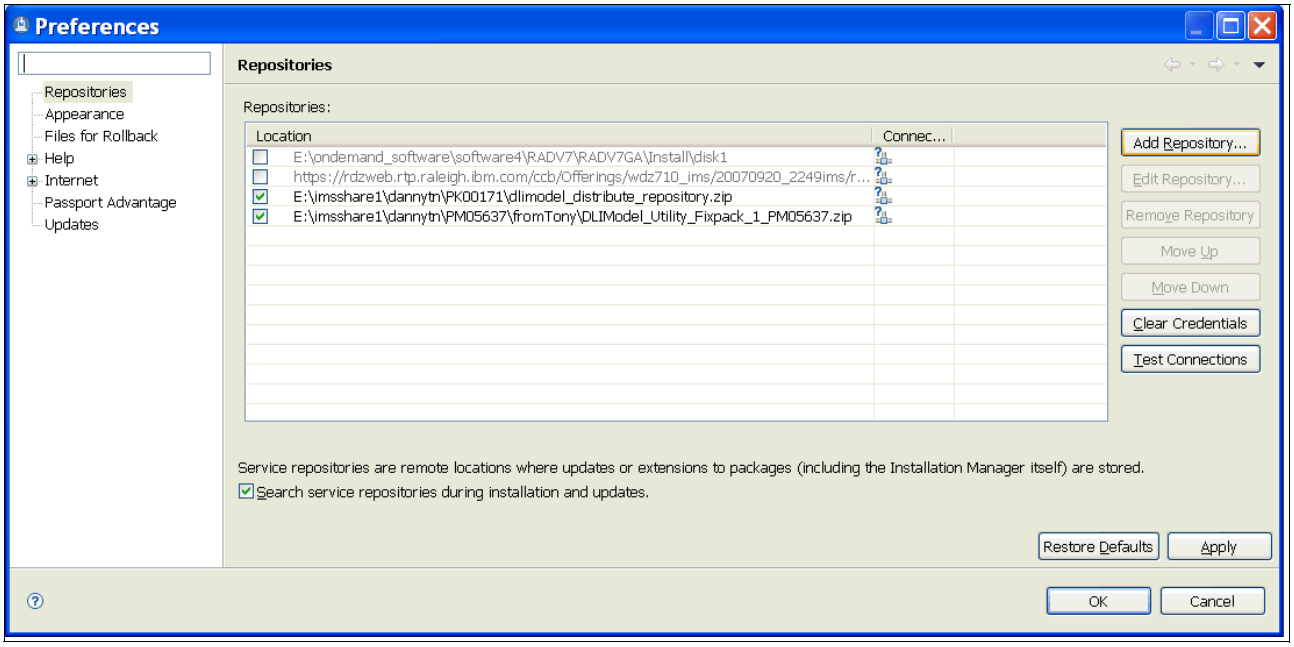


Figure 4-10 Adding fix pack to the IBM Installation Manager Repository

2. After you add the fix pack, you can apply the maintenance. Click **Update**, and select the package group where the DLI Model Utility is installed, and then follow the windows.
3. If you want to install the DLIModel Utility directly with the fix pack, select **Check for Other Versions and Extensions** during the installation, as shown in Figure 4-9 on page 86. The Installation Manager shows you the latest Version and directly installs the newest version.

Shell sharing

During the installation process, notice that one of the panels includes an option that enables you to extend an existing Eclipse IDE or JVM, as shown in Figure 4-11 on page 88. Do not select this option unless you are already using Eclipse and simply want to add the Eclipse-based DLIModel utility features to your existing environment. In the latter case, select the option, specify the location of your Eclipse.exe, and continue to follow the prompts.

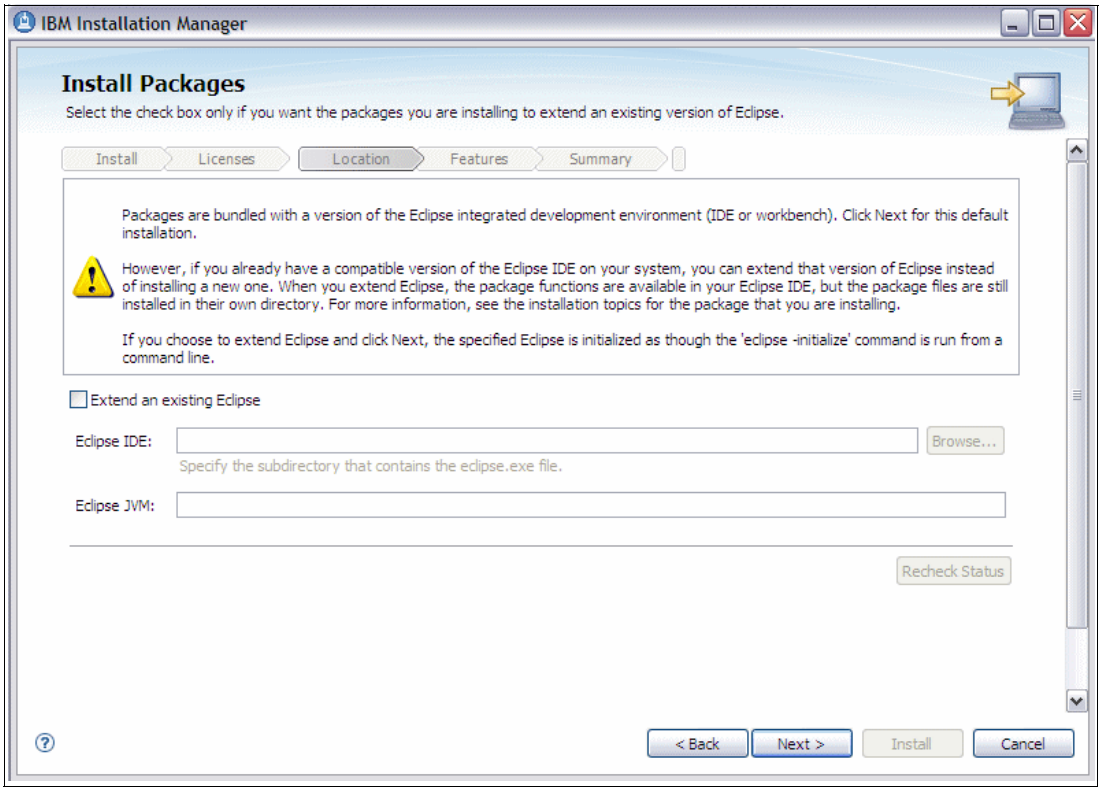


Figure 4-11 Option for the DLIModel utility to shell share with other Eclipse products

The ability to install products that share a common environment is provided by the IBM Installation Manager and is known as shell sharing, which has several advantages. You can consolidate separate product features into one user interface, eliminating duplicate installations of the common environment on your machine. Not only does this save disk space and installation time, but it allows all products that share the same shell to receive updates when they are installed by the IBM Installation Manager. Refer to section 4.6.3, “Data type conversion table” on page 96 for an alternate way to extend an existing Eclipse IDE.

Restriction: For the DLIModel utility plug-in to shell share with another Eclipse-based product, they both must use the same version of Eclipse. Additionally, the features that are associated with each product must not conflict with each other.

4.4 Setting up for sample scenarios included in this book

To be able to follow the scenarios presented in the other chapters in this book, there are a few required prerequisite set up steps. These steps include downloading the sample database source and using the DLIModel Utility plug-in to generate and edit its metadata, which you ultimately export as a jar file for use with Java application development. We begin with describing how to download the database source.

4.4.1 Downloading the Car Dealer IVP database source

The IMS product includes a sample database called Car Dealer. The source that is associated with this database is included in the IVP and is located in the *HLQ*.SDFSISRC

data set. In our case, the *HLQ* is IMS11B, so the data set we use is named IMS11B.SDFSISRC.

You have several options to copy the database source from the host to your workstation. You can use the functions of your 3270 terminal emulation, or you can use Rational Developer for System z to easily access the members in this data set. You can either copy/paste the members or use FTP to download them. In the following steps we show an example of downloading the data set members with the Microsoft Windows XP integrated FTP client:

1. Click **Windows Start** → **Run**, and type `cmd.exe` to open the command prompt.
2. Start the FTP session using the command FTP <HOSTIP>, and when prompted, insert your TSO user name and your password. After login, enter the following commands:

```
cd  'IMS11B.SDFSISRC'
lcd c:\temp to change to an existing directory on your system
ASC to set transfer mode to ascii
get DFSAUTDB AUTODB.dbd (for the DBD)
get DFSEMPDB EMPDB2.dbd (for the DBD)
get DFSEMLDB EMPLDB2.dbd (for the Logical DBD)
get DFSLAUTO AUTOLDB.dbd (for the Logical DBD)
get DFSIND22 SINDEX22.dbd (for Index DBD)
get DFSIND11 SINDEX11.dbd (for Index DBD)
get DFSAUT11 AUTPSB11.psb (for PSB)
```

3. To exit the FTP session, type `quit`, and to end the command prompt type `exit`.

You now have the source members in your local directory, in this case c:\temp. You must rename the source members to match the DBD name statements that were included in the source data. To do this, use the **get** command by specifying **get <sourcename on the current remote directory> <destination name on the current local directory>**.

For a detailed explanation of the Car Dealer database and its source code, refer to Appendix B, “Car Dealer IVP database” on page 237. Now that you obtained the sample database source files, you can use the DLIModel utility plug-in to generate metadata for it, which we now discuss.

4.5 Using the IMS Enterprise Suite DLIModel utility

To use the DLIModel utility plug-in to generate metadata and other artifacts that Java can understand, we use the IMS source files obtained in the previous section as input. In this section, we discuss using the DLIModel wizard and DLIModel editor to generate and edit metadata that is associated with the sample database (respectively) and exporting this metadata as a jar file.

When working with the DLIModel utility, all associated data is contained within a DLIModel project, which you must first create. The DLIModel wizard assists you in creating a project and gives you the option of creating an XML schema, generating a result report, and writing a trace log. If you already have a DLIModel project that you are working with, you can import IMS PSB and DBD source files into it.

After you generate the metadata, you can use the DLIModel editor to modify it and also import information from COBOL or PL/I copybooks. Also, you can use the editor to search, save, and print the hierarchy of the IMS database.

4.5.1 Generating metadata for Car Dealer database

In this section, we examine how to use the DLIModel utility to generate the Java Metadata class file, which is required in the scenario use cases in this book. A prerequisite to this step is downloading the required source files, which we covered in 4.4, “Setting up for sample scenarios included in this book” on page 88.

After you launch the Eclipse-based IMS Enterprise Suite DLIModel Utility, follow these steps to generate metadata for our Car Dealer sample database (which is included in the IVP):

1. Click **File** → **New** → **Projects**, select **DLIModel Utility Project**, and click **Next**.
2. Specify `AUTPSB11` for the project name and `samples.ims.openDb` for the Java package name. If you show the Advanced section by clicking the button, you can then select additional options for the generation, including the ability to use an IMS control statement from the former version of the DLIModel utility to generate the Metadata Class file (see Figure 4-12).

Important: Be aware that the Java package is case-sensitive and is referenced in all scenarios in this book as the package and class name shown here.

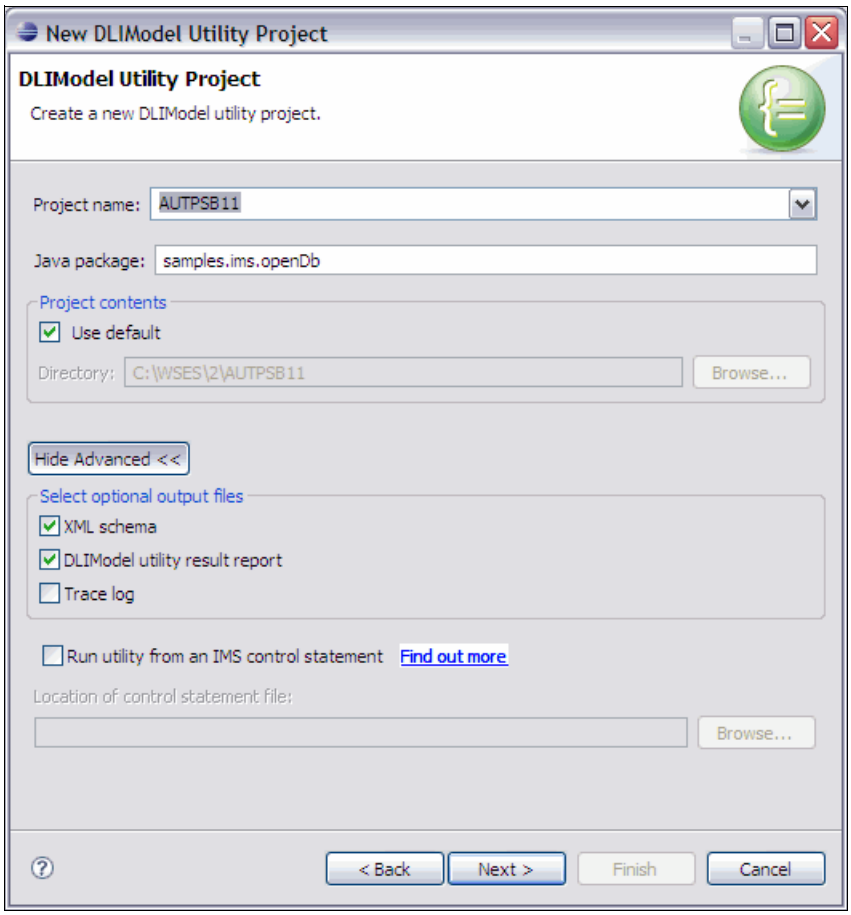


Figure 4-12 New DLIModel Utility Project - Step 1

3. Leave the defaults as they are, and click **Next**.
4. Highlight your local directory, and select the downloaded AUTPSB11.psb file, as shown in Figure 4-13 on page 91. Since the DBD source files are in the same directory and will be

named as defined in the PSB's DBDNAME statement, it automatically finds and selects the associated DBDs. Otherwise, you must manually select all of the DBD source members that are associated with the databases and indexes. These members can have either a file extension of .dbd or .psb or none.

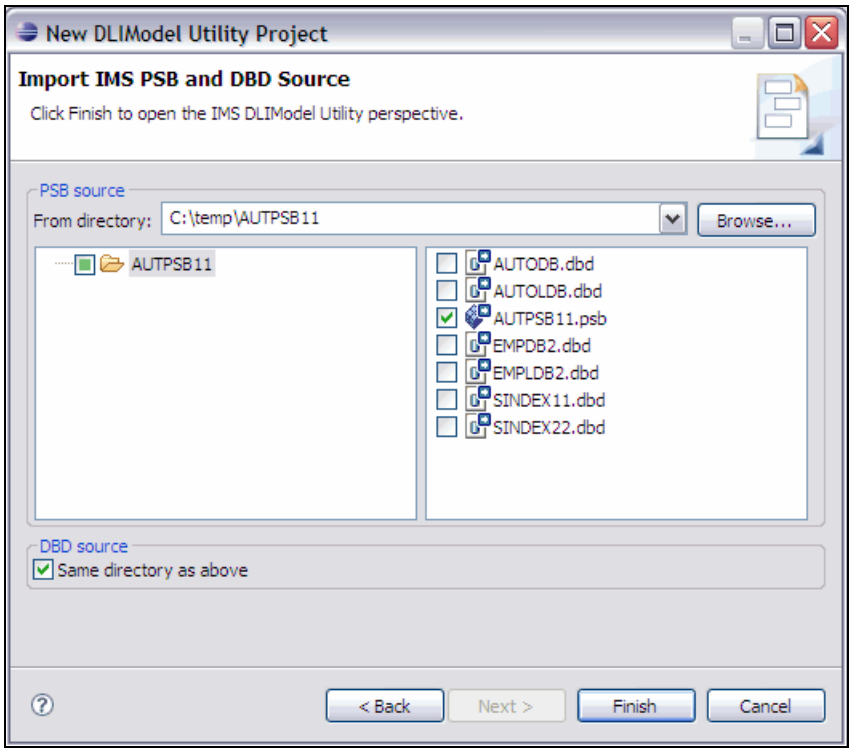


Figure 4-13 New DLIModel Utility Project - Step 2

5. Click **Finish**, and you will see the generated files in the Package Explorer, as shown in Figure 4-14.

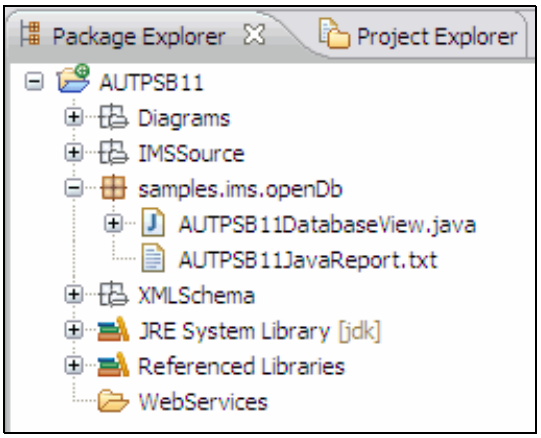


Figure 4-14 New DLIModel Utility Project - Step 3

6. The required file is in the samples.ims.openDb Java package and is automatically called AUTPSB11DatabaseView.

4.5.2 Editing the AUTPSB11 Project

Since the DLIModel utility cannot obtain all of the required data from the PSB and DBD source files, you must review, and if necessary, edit the files that were generated.

After generating the files that are now contained in the Project, the Overview diagram automatically opens itself, as shown in Figure 4-15. Notice that at the bottom of this view, you can switch between PCBs that are associated with this PSB. Any saved changes that you make in the diagram will result in the regeneration of the files in the project.

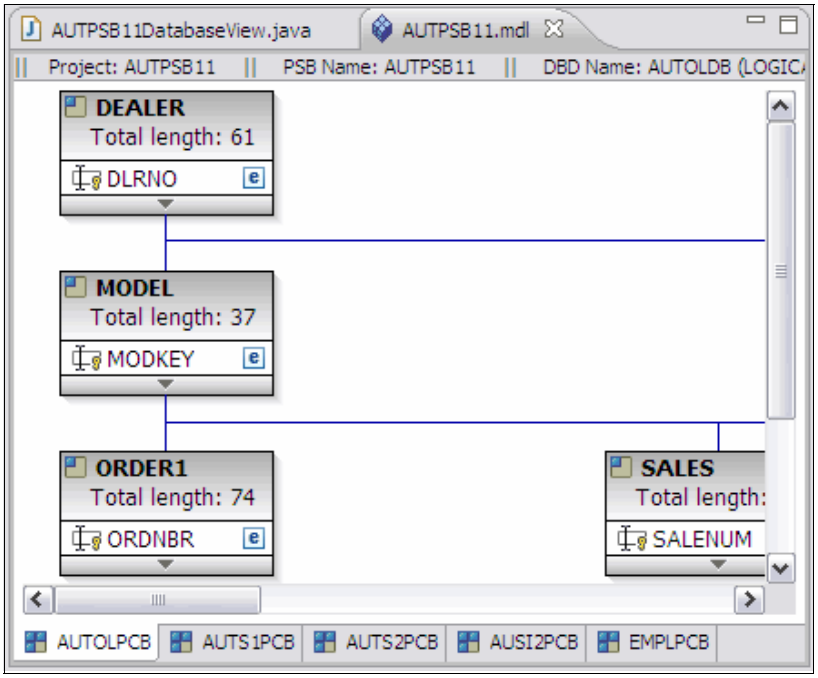


Figure 4-15 AUTPSB11 Overview diagram

When you look at the definition of the various fields, you might notice that the fields are all defined as Character Data Type because IMS does not have a large variety of data types. The only differentiation that the DBDs and PSBs make concerning the data type is in the FIELD parameter and can be any of the following items:

- X stands for Hexadecimal data
- P stands for Packed decimal data
- C stands for Alphanumeric data or a combination of types of data
- F stands for Binary fullword data (only MSDB)
- H stands for Binary halfword data (only MSDB)

However, it can be important to change the data type from Character to another data type, for example, if it is a Packed decimal field in IMS. This particular field format is normally handled by the application in IMS, for example, the Packed decimal field that needs five bytes can contain more than five numbers.

The format of these fields is normally specified by the IMS application; therefore, you have two options for correcting the data types:

- Use the COBOL copybook import function to get the required data out of the copybook. Right-click the diagram, and select the Import Copybook Fields, as shown in Figure 4-16.

> **Note:** The import COBOL copybook feature only works if your DLIModel Utility is running as a plug in to Rational Application Developer or Rational Developer for System z.

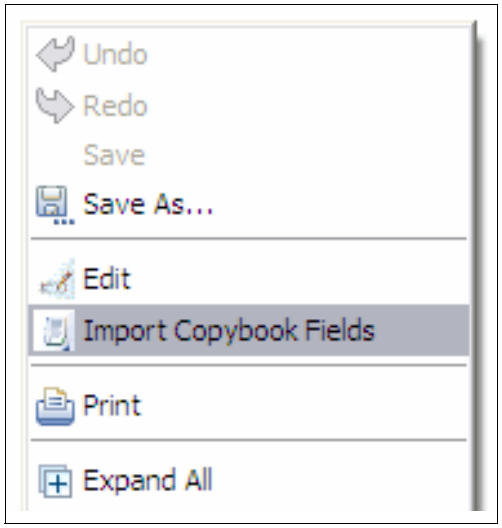


Figure 4-16 Import Copybook Fields

- The second option is to specify the values manually in the diagram by right-clicking a field and, select **Edit**.

In the Project's IMSsource folder of the Package Explorer, open AUTODB.dbd. When you have a look at the types, you will find the following fields, which are not TYPE=C, in the ORDER segment:

```
FIELD NAME=MSRP,BYTES=5,START=27,TYPE=P
FIELD NAME=COUNT,BYTES=2,START=32,TYPE=P
```

You will find the following line in the STOCK segment:

```
FIELD NAME=WRNTY,BYTES=1,START=46,TYPE=X    NEW (BOOLEAN)
```

You have two options for this example.

- Leave the fields as CHAR (recommended) because they are currently filled with CHAR data by the IVP jobs.
- You can also delete the contents in the database manually (for example, with a DFSDDLT0 job), and adjust the types in the diagram. Because there is no copybook provided for this IVP sample database, you must adjust them manually. To do this, right-click in the diagram on one of the values, and click **Edit** or use the Properties view to edit the fields. Specify `MSRP` as data type PACKEDDECIMAL with the representation of `S9999999V99` and `COUNT` as data type PACKEDDECIMAL with the representation of `S999`. Change the WRNY field to `BIT` because it is used here only as a Boolean field with two hexadecimal values.

Attention: The MSRP and COUNT segments are currently loaded by the IVP with a few values in CHAR format that are not PACKEDDECIMAL, as defined. So the type conversion does not work if you specify these fields as PACKEDDECIMAL.

4.5.3 Exporting the metadata as a Jar file

After editing the Project, you can now export it by following these steps:

1. Click **File** → **Export**, select **Java** → **Jar file**, and click **Next**.
2. By default the whole project is selected for export. Clear the diagram and the WebService folder. Select the **Export Java Source files and resources** option to include the source files in your Jar file. This way, when you provide the Jar file to your application developer, he or she can review the generated Java Report and the source code used to generate the files. Also, if you have future database changes, you can easily identify the correct version of the Jar file. Type in the path and the name where the Jar file is to be exported. In Figure 4-17, you can see that we specified c:\temp\AUTPSB11.jar.

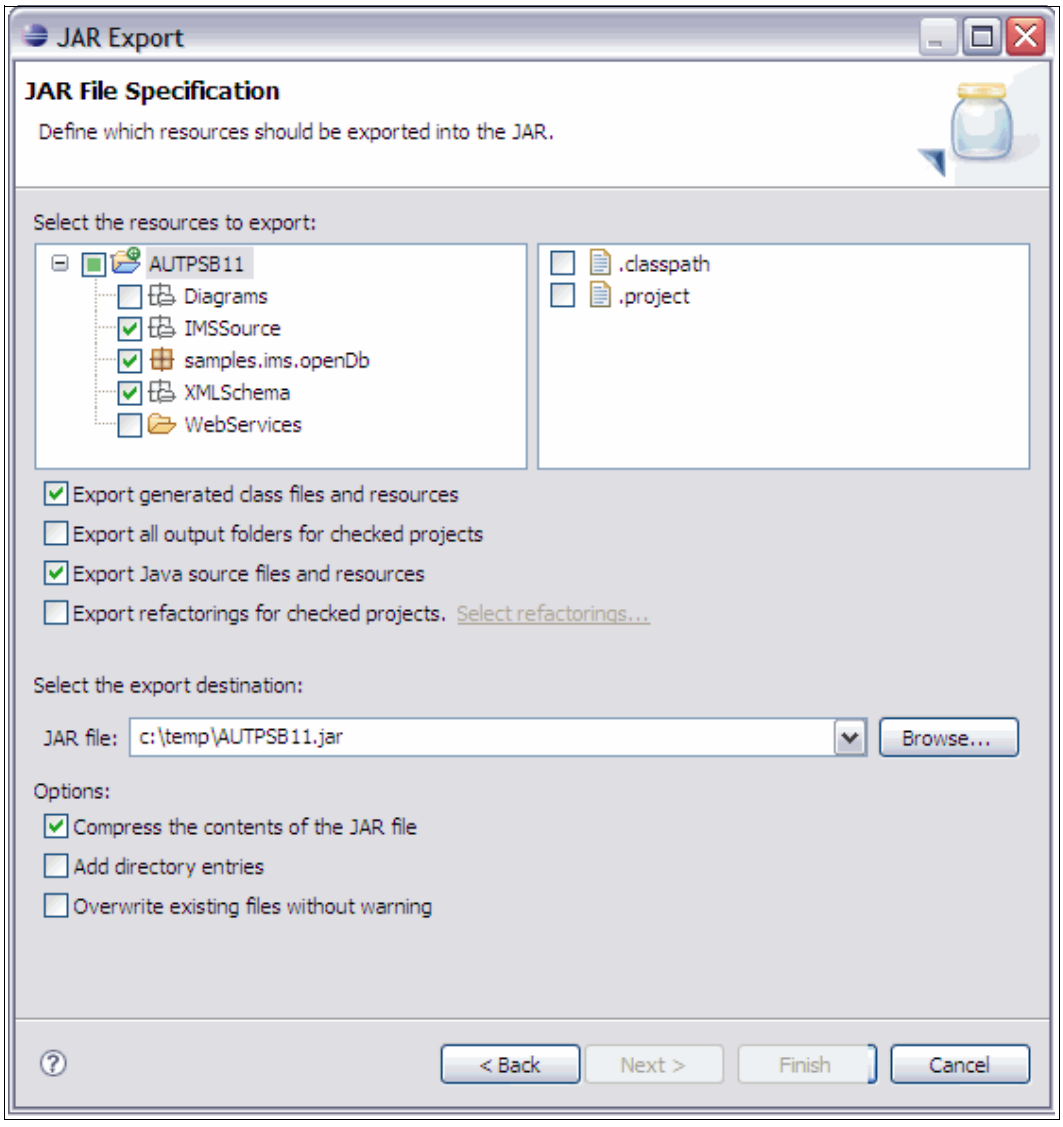


Figure 4-17 Export of AUTPSB11.jar

3. Click **Finish** to complete the Export.

4.5.4 Integrating the DLIModel Utility with other Eclipse products

If you want to use the DLIModel Utility without installing it on every PC, one option is copying the plug-in to the Eclipse plug-in folder. As a reminder, the Eclipse installation must have the minimum requirements to run the DLIModel Utility plug-in. If you did not originally use the IBM Installation Manager to install Eclipse, we recommend that you manually check for updates associated with newer Eclipse versions. To copy the plug-in manually:

1. Copy the com.ibm.ims.dlimodel.ui_2.0.3 directory from your installation directory (whose default is C:\Program Files\IBM\IMS Enterprise Suite V1.1\DLIModel Utility Plug-in\plugins) to your target Eclipse plug-ins directory. If you are using Rational Developer for System z, version 7.6, the default is C:\Program Files\IBM\SDP\plugins.

Eclipse: With newer Eclipse versions you do not have to issue an Eclipse **-clean** command to load the plug-in. Instead, simply restart your Eclipse to begin using it.

4.6 Additional considerations for the IMS Enterprise Suite DLIModel Utility

There are several other considerations for using the IMS Enterprise Suite DLIModel Utility, which we discuss in this section.

4.6.1 Ensuring consistency between generated class files and other JRE files

The current Eclipse version of the IMS Enterprise Suite DLIModel utility uses JDK 6.0 by default to generate the metadata class files. If you plan to work with other files that are based on a separate Java Runtime Environment (JRE), for example, version 1.5, you must change the version to match before compiling a java application. If you do not do this, you will receive a *Java class version* error when you compile. To change the version:

1. Right-click your project, and select **Properties**.
2. Select the **Java Compiler** page, and select the **Enable Project specific settings** option. Select your **Compiler compliance level** to the level you need. In Figure 4-18 on page 96 it is specified as 1.5.

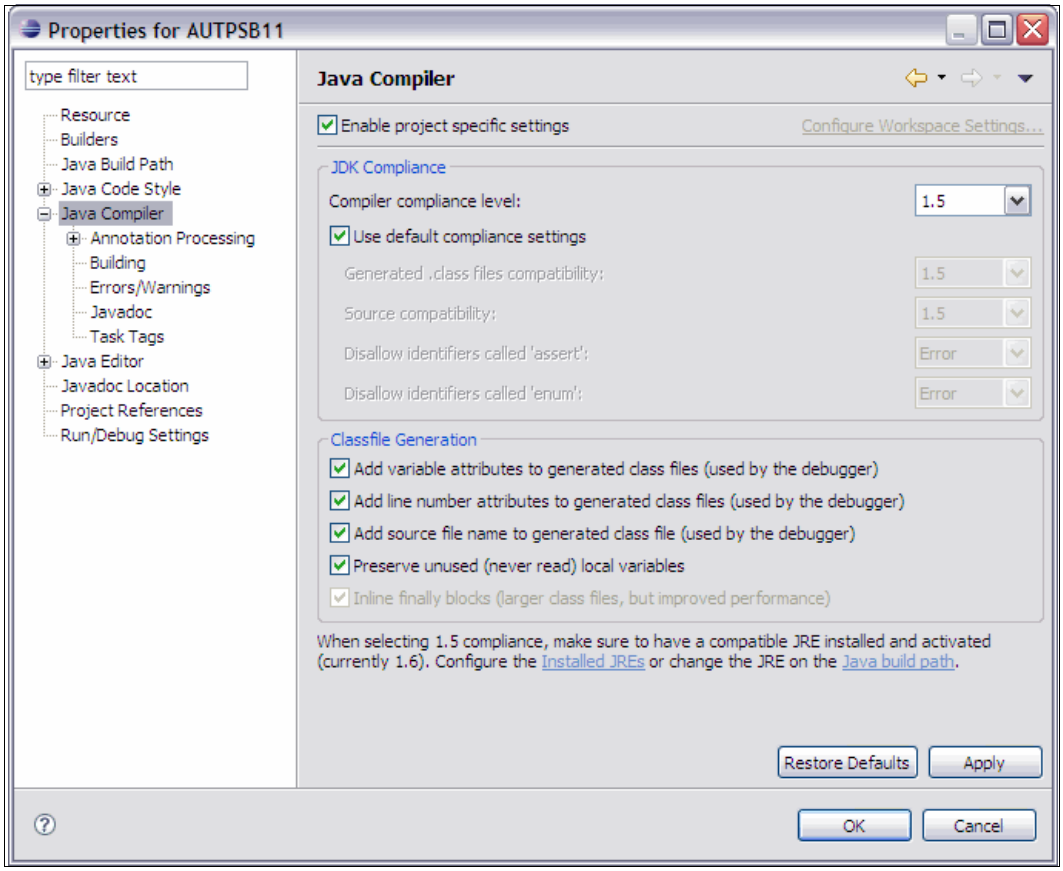


Figure 4-18 Compiler compliance level

3. You will be asked to regenerate the project. Click **Yes** to rebuild it. Make sure that you export your Jar file again to be able to use it.

4.6.2 Tracking changes to IMS database definitions

If you change your IMS database design and redefine the database, you must also recreate the metadata files for this database, and consider this part of the change management process of your company. If you do not have the source of your DBDs and PSBs, you can use IMS Tools, such as the *IBM IMS Library Integrity Utilities for z/OS* to recover it from the IMS libraries, which also guarantees that you have the most current source for your database definitions and PSBs.

4.6.3 Data type conversion table

Table 4-1 shows a mapping of COBOL copybook data definitions and how they can be mapped by the DLIModel Utility.

Table 4-1 Conversion table: Copybook format to data types

Copybook format	DLIType info data type	Java data types
PIC X(25)	CHAR	java.lang.String
PIC S9 (1-4 figures) COMP	SMALLINT (2 Bytes)	short
PIC S9 (5-9 figures) COMP-4	INTEGER (4 Bytes)	int

Copybook format	DLIType info data type	Java data types
PIC S9 (10-18 figures) BINARY	BIGINT (8 Bytes)	long
COMP-1	FLOAT	float
COMP-2	DOUBLE	double
PIC S9(06)V99 COMP-3	PACKEDDECIMAL	java.math.BigDecimal
PIC S9(06)V99	ZONEDECIMAL	java.math.BigDecimal
PIC 9(06).99	ZONEDECIMAL	java.math.BigDecimal
PIC 9 DISPLAY	ZONEDECIMAL	java.math.BigDecimal

5

IMS Open Database for application developers

In this chapter, we provide an overview of the IMS Version 11 Open Database functions and how to use them for developing applications.

This chapter is intended for Java application programmers who want to get an understanding of what the IMS Version 11 Open Database feature can provide for developing applications, the components involved, and their requisites.

In this chapter we discuss the following topics:

- Overview of IMS Open Database on the application side
- Architectural considerations
- IMS Universal Database resource adapter
- IMS Universal JDBC driver (stand-alone)
- IMS Universal DL/I driver
- SQL syntax for the IMS Universal drivers

5.1 Overview of IMS Open Database on the application side

IMS Version 11 makes it much easier to get to data that is stored in the IMS Databases. The Open Database feature of IMS Version 11 enables you to take advantage of your existing valuable IMS data by reusing it easily in new applications.

IMS provides drivers for multiple environments and programming paradigms. The exciting thing about this drivers are that they are for the first time possible to access IMS data out-of-the-box without any additional Tool or Product from distributed clients. It also does not matter whether the application is running on the z/OS mainframe or it is running on any other distributed platform, which helps your company to keep costs down for additional Tools and Products and get rid of additional steps, such as replicating the data to an accessible place.

For developing a Java application to access IMS data, you need your Integrated Development Environment (IDE), one of the IMS Universal drivers, and the Database metadata for the database that you want to access in IMS.

5.1.1 IMS Universal DB drivers

The IMS Universal DB drivers are a set of Java classes and resource adapters that enable the access to IMS databases. They are built on industry standards and open specifications. The IMS Universal DB drivers can communicate remotely using TCP/IP (type-4 connectivity) and locally (type-2 connectivity).

The type-4 connectivity is based on the Distributed Relational Database Architecture (DRDA) protocol, but you do not need to know how to program with DRDA if you are using Java as the programming language. If you are using the type-4 connectivity, you need IMS Connect on the z/OS side, which is an integral part of IMS. IMS Connect acts as the TCP/IP endpoint for the IMS Universal drivers.

The IMS Universal DB drivers also have a local option (type-2 connectivity) included. You can use this type of connectivity for accessing IMS data when your application program is running on the same logical partition (LPAR) as your IMS. For the local connectivity, you do not need IMS Connect because TCP/IP is not used for this direct communication.

The drivers have multiple implementations that you can use based on your application environment and your application architecture. They provide an application programming framework that offers the greatest choice of options for accessing IMS data. These programming options are:

- IMS Java dependent region resource adapter

  The IMS Java dependent region resource adapter is a set of Java classes and interfaces that support IMS database access and IMS message queue processing within Java batch processing (JBP) and Java message processing (JMP) regions. It provides the same DL/I functionality that is provided in message processing program (MPP) and non-message driven BMP regions.

- IMS Universal DB resource adapter

  A Java EE Connector Architecture (JCA) 1.5-compliant resource adapter for your managed JEE Environment. It provides access to IMS data using the Common Client Interface (CCI) and Java Database Connectivity (JDBC) interfaces.

- IMS Universal JDBC driver

  A stand-alone Java Database Connectivity (JDBC) driver that implements the JDBC 3.0 API for making SQL-based database calls against IMS Databases.

- IMS Universal DL/I driver

  A Java API for making calls with traditional DL/I programming semantics.

The IMS Universal drivers are installed as part of the IMS Installation Procedure (SMP/E) as an optional part (FMID JMK1106). They are maintained using the normal maintenance process of IMS. The IMS Universal drivers are mounted in the z/OS OMVS file system and can be downloaded using FTP from your mainframe. Usually they are mounted to the following path:

```
/usr/lpp/ims/ims11/imsjava/
```

The exact path to download the files can vary depending on your system configuration. Ask your IMS System programmer for help on how to get the drivers.

The Universal DB Resource Adapter is shipped as four rar files:

- imsudbJLocal.rar

  Contains the local transaction capable driver for JDBC access
- imsudbJXA.rar

  Contains the two-phase commit capable driver for JDBC access
- imsudbLocal.rar

  Contains the local transaction capable driver for JCA CCI access
- imsudbXA.rar

  Contains the two-phase commit capable driver for JCA CCI access

The JDBC and DL/I drivers are shipped in a single jar file, imsudb.jar.

Note: When you download the IMS Universal drivers using FTP, download them in binary mode.

5.1.2 IMS database metadata

IMS has its own catalog-like library called ACBLIB. Within ACBLIB all information about the IMS application's view on the database that is contained within the Program Specification Block (PSB) and the physical layout of the database that is contained within the Database Definition (DBD) is stored.

The ACBLIB is currently only accessible by IMS itself and not by external applications. But the IMS Universal DB driver must know how the database layout when you want to access it from your application. Therefore you need metadata about the database that you want to access in your application.

The metadata files are Java class representations of the IMS Database View that is generated from the PSB and DBD sources from IMS. This step occurs with the help of the IMS Enterprise Suite, as we discussed in Chapter 4, "Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility" on page 77. Metadata must generated by your IMS System programmer because he has access to the necessary resources in IMS.

The metadata Java class maps the hierarchy of the segments to Java arrays of the Type DLITypeInfo. These arrays contain the fields and define the data type of each column, the column length, and offset within the segment. Example 5-1 on page 102 shows an extract of the metadata Java class from our Car Dealer IVP Example.

Example 5-1 Extract of the Java metadata class from the Car Dealer IVP Example

```
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class AUTPSB11DatabaseView extends DLIDatabaseView {
// This class describes the data view of PSB: AUTPSB11
    // PSB AUTPSB11 has database PCBs with 8-char PCBNAME or label:
    // AUTOLPCB,AUTS1PCB,AUTS2PCB,AUSI2PCB,EMPLPCB

    // Constructor
    public AUTPSB11DatabaseView() {
        super("2.0.3","AUTPSB11", "AUTOLPCB", "AUTOLPCB", AUTOLPCBarray, "AP");
        addDatabase("AUTS1PCB", "AUTS1PCB", AUTS1PCBarray, "GRP");
        addDatabase("AUTS2PCB", "AUTS2PCB", AUTS2PCBarray, "GRP");
        addDatabase("AUSI2PCB", "AUSI2PCB", AUSI2PCBarray, "GRDP");
        addDatabase("EMPLPCB", "EMPLPCB", EMPLPCBarray, "AP");
    } // end AUTPSB11DatabaseView constructor

    // The following describes Segment: DEALER ("DEALER") in PCB: AUTOLPCB ("AUTOLPCB")
    static DLITypeInfo[] AUTOLPCBDEALERArray= {
        new DLITypeInfo("DLRNO", DLITypeInfo.CHAR, 1, 4, "DLRNO", DLITypeInfo.UNIQUE_KEY),
        new DLITypeInfo("DLRNAME", DLITypeInfo.CHAR, 5, 30, "DLRNAME"),
        new DLITypeInfo("CITY", DLITypeInfo.CHAR, 35, 10, "CITY"),
        new DLITypeInfo("ZIP", DLITypeInfo.CHAR, 45, 10, "ZIP"),
        new DLITypeInfo("PHONE", DLITypeInfo.CHAR, 55, 7, "PHONE")
    };
    static DLISegment AUTOLPCBDEALERSegment= new DLISegment
        ("DEALER","DEALER",AUTOLPCBDEALERArray,61);

    // The following describes Segment: MODEL ("MODEL") in PCB: AUTOLPCB ("AUTOLPCB")
    static DLITypeInfo[] AUTOLPCBMODELArray= {
        new DLITypeInfo("MODTYPE", DLITypeInfo.CHAR, 1, 2, "MODTYPE"),
        new DLITypeInfo("MODKEY", DLITypeInfo.CHAR, 3, 24, "MODKEY",
DLITypeInfo.UNIQUE_KEY),
        new DLITypeInfo("MAKE", DLITypeInfo.CHAR, 3, 10, "MAKE"),
        new DLITypeInfo("MODEL", DLITypeInfo.CHAR, 13, 10, "MODEL"),
        new DLITypeInfo("YEAR", DLITypeInfo.CHAR, 23, 4, "YEAR"),
        new DLITypeInfo("MSRP", DLITypeInfo.CHAR, 27, 5, "MSRP"),
        new DLITypeInfo("COUNT1", DLITypeInfo.CHAR, 32, 2, "COUNT")
    };
    static DLISegment AUTOLPCBMODELSegment= new DLISegment
        ("MODEL","MODEL",AUTOLPCBMODELArray,37);
...
```

Example 5-1 shows that the Java class contains all segments with their fields as Java objects. The metadata class defines the type, the length, and the name of the columns. You can manually change these attributes if you have special requirements.

Note: You can see that the column COUNT in the MODEL segment was automatically assigned the alias COUNT1 because Count is a restricted SQL keyword.

5.1.3 Java version requirements

Java application programs that use the IMS Universal drivers require at least the Java Development Kit 5.0 (JDK 5.0). Java programs that run in IMS Java dependent regions require the Java Development Kit 6.0 (JDK 6.0) or later.

5.2 Architectural considerations

The IMS Universal drivers have several implementations that you can choose from. Which driver you choose for application development and which options you set depends on several factors, which we examine in the following sections.

5.2.1 Transactional support

You must decide which level of transactional support you need in your application. JCA differentiates between drivers with Global Transaction support and Local Transaction support.

Global Transaction support (XA)

The XA standard is an X/Open specification for distributed transaction processing (DTP). It describes the interface between the global transaction manager and the local resource manager. The XA specification describes what a resource manager must do to support transactional access. On the JEE platform, the XA specification is driven through the transactional system contract. The interface that the contract uses is the XAResource interface. A resource adapter that is XA compliant must implement this interface. At transaction commit time, the resource managers are informed by the transaction manager to prepare, commit, or rollback a transaction according to the two-phase commit protocol. There are also one-phase optimizations built into the XAResource system contract.

The X/Open XA Protocol ensures that all components in a transaction either commit or rollback their changes for one transaction, which occurs using the 2 Phase Commit Protocol with all used resources. A higher instance, usually an application server, interacts as Sync Point Manager. It controls each connection and executes the necessary 2 Phase Commit functions for each resource in the responsible driver. The driver must implement the necessary functions for the two-phase commit handling.

If you choose this option with the IMS Universal drivers, this means that Recovery Resource Services (RRS) are used on the z/OS side, and the JEE Container takes the role of Sync point Manager across all connections to z/OS and the distributed environment. IMS Connect builds the necessary RRS structure to support the two-phase commit protocol. The XA supporting drivers have a detection for one-phase commit processing, if no two-phase processing is necessary.

Local transaction support

With Local Transaction support, each connection to a resource runs in its own transactional unit of work, giving you the ability to decide where you want to set a commit point or rollback of the transaction. But this means also that you must take care of the consistency of your data on your own, which can be very complex, especially when you have an application that connects to IMS and other resources, such as DB2, and you do not have global transaction support in place across the various drivers and connections. You must commit and rollback each unit of work individually and take care of data integrity in case of application failures.

Local Transaction support can be useful if you have an application that only connects to one resource. It can be faster because the driver does not have to handle the two-phase commit functions. It can also be useful if you do not want to use RRS on the z/OS side or if you have a need for individual handling of the commit-points.

5.2.2 Access types

Two major types of connectivity are supported by the IMS Universal drivers:

- Local connectivity to IMS databases on the same Logical Partition as IMS (type-2 connectivity)
- Distributed connectivity through TCP/IP from anywhere (type-4 connectivity)

Distributed access (type-4 connectivity)

With distributed access, the IMS Universal drivers can run on any platform that supports TCP/IP and a Java Virtual Machine (JVM), including z/OS. This feature is referred to as type-4 Connectivity. Type-4 drivers are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

The IMS Universal drivers first establish a TCP/IP-based socket connection to IMS Connect on a certain Port. IMS Connect routes the request to the IMS databases using the Open Database Manager and sends the response back to the client application.

The IMS Universal drivers support connection pooling with type-4 connectivity, which limits the time that is needed for allocation and deallocation of TCP/IP socket connections. To maximize connection reuse, only the socket attributes of a connection are pooled. These attributes include the IP address and port number that the host IMS Connect is listening on. As a result, the physical socket connection can be reused and additional attributes can be sent on this socket to connect to an IMS database.

Figure 5-1 on page 105 shows how the IMS Universal drivers route communications between your Java client applications that run in a distributed environment and an IMS subsystem using type-4 connectivity.

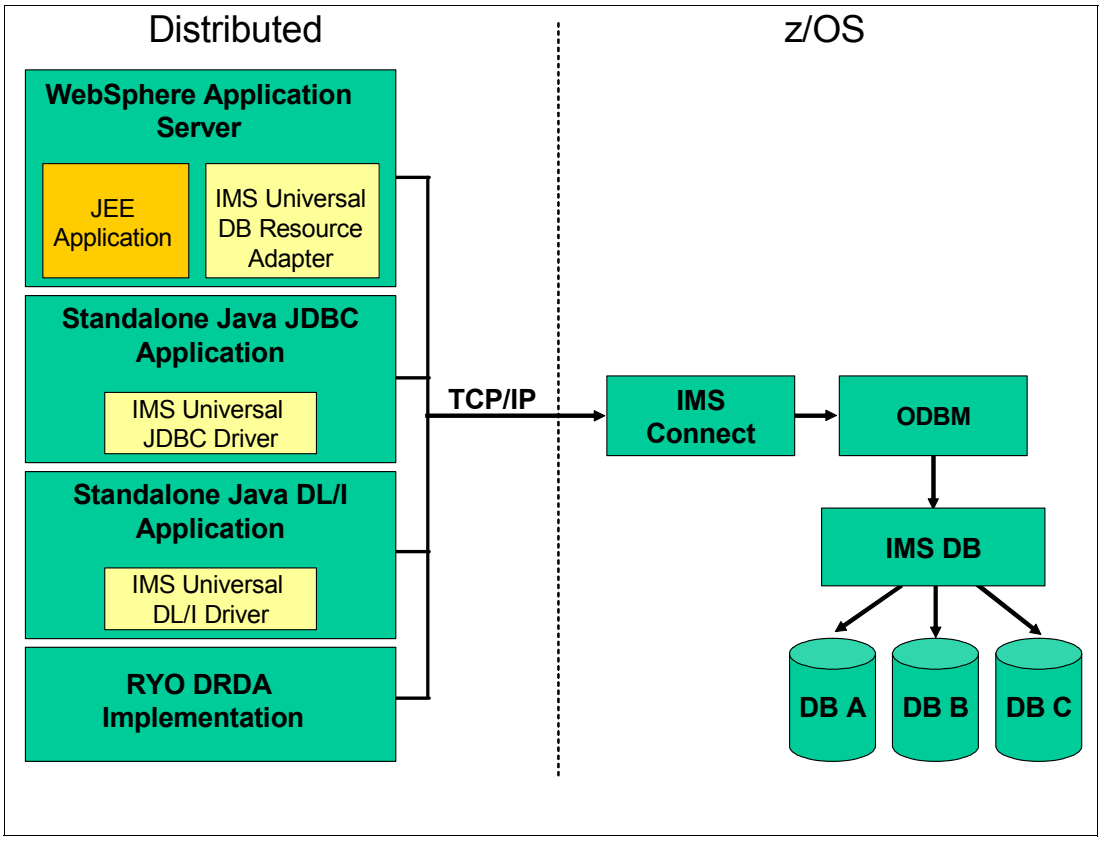


Figure 5-1 Distributed access - Type-4 connectivity

You can also use the IMS Universal drivers with type-4 connectivity, if your Java clients are running in a z/OS environment but are located on a separate logical partition from the IMS subsystem. Use type-4 connectivity in a z/OS environment if you want to isolate the application runtime environment from the IMS subsystem environment.

Local access (type-2 connectivity)

With local access, the application that uses the IMS Universal drivers must be on the same LPAR that IMS is on. This access is referred to as type-2 connectivity. This type of access does not need IMS Connect because it communicates directly locally instead through TCP/IP.

Type-2 drivers are written partly in the Java programming language and partly in native code. The drivers use a native client library that is specific to the data source to which they connect. Because of the native code, their portability is limited.

Table 5-1 shows the z/OS runtime environments that support client applications using the IMS Universal DB drivers with type-2 connectivity.

Table 5-1 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity

z/OS runtime environment	Recommended IMS Universal type-2 driver	Used component
WebSphere Application Server for z/OS	► IMS Universal DB resource adapter	ODBA, DRA
CICS Java Applications	► IMS Universal JDBC driver ► IMS Universal DL/I driver	DRA

z/OS runtime environment	Recommended IMS Universal type-2 driver	Used component
IMS Java dependent regions (JMPs and JBPs)	► IMS Universal JDBC driver ► IMS Universal DL/I driver ► IMS TM Resource Adapter	IMS native access

Each of the products (CICS, DB2, IMS) has an integrated type-2 interface, which will be used by the IMS Universal drivers' RRSLocalOption connectivity type.

In addition to type-4 and type-2 connectivity, the RRSLocalOption connectivity type is supported by the IMS Universal DB resource adapter running on WebSphere Application Server for z/OS. With RRSLocalOption connectivity, applications using the IMS Universal DB resource adapter do not issue commit or rollback calls; instead, WebSphere Application Server for z/OS manages transaction processing.

Enabling the connection type: Set the driverType on the connection settings to the value `RRSLocalOption` to enable this connection type.

IMS Universal driver settings

The settings in Table 5-2 can be used with the IMS Universal drivers, depending on your access type.

Table 5-2 Settings for the IMS Universal drivers

Setting and Description	Type-4	Type-2 / RRSLocalOption
driverType Defines the type of access whether TCP/IP or within LPAR	4 or IMSManagedConnectionFactory.DRIVER_TYPE_4	2 / RRSLocalOption
datastoreName Defines the IMS to connect to	IMS ID (or Alias) configured in the ODBM Configuration or empty for IMSplex	IMS ID specified in the DRA table used for accessing IMS
metadataURL Defines the fully qualified classname of the metadata generated through the DLIModel Utility	class://samples.PSBDatabaseView	class://samples.PSBDatabaseView
datastoreServer Defines the endpoint for the IMS Universal driver	IP or host name of IMS Connect	-
portNumber Defines the port specified on the ODACCESS parameter of IMS Connect	5555 (Default)	-
user Specifies the UserID	RACF User ID	-
password Specifies the User password	RACF Password	-

5.2.3 Programming approach

There are multiple programming approaches for the IMS Open Database feature with separate capabilities and access methods. The overview in this section can help you to find the right programming approach for your application's requirements.

JCA/JDBC with SQL

The JCA/JDBC programming model eases your life in a managed Application Server environment because it uses standard JDBC methods with standard SQL syntax and makes use of the management capabilities of the environment using the JCA. With the XA supported IMS Universal drivers, the Application Server takes care of the two-phase commit handling between JCA drivers in your application. The big advantage of this approach is that JDBC and SQL are usually very well known by application developers because they are used by many database drivers. JDBC is the industry standard for database-independent connectivity between the Java programming language and any database that implemented the JDBC interface. The client uses the interface to query and update data in a database. It is the responsibility of the JDBC driver itself to implement the underlying access protocol for the specific database that the driver is implemented for.

JCA/CCI with SQL or DL/I

The Common Client Interface (CCI) is the other approach for managed Application Server environments and is part of the JCA specifications. The advantage of this approach is that you can use SQL syntax and also DL/I commands within one unit of work to get the maximum out of your IMS database and at the same time use the management capabilities of the Application Server. The disadvantage of this approach is that the CCI model is more difficult to understand and to handle. Besides the complexity of the CCI model, this approach can help you to realize applications with special data processing requirements, such as batch processing of data in a sequential hierarchical order fusing DL/I calls followed by SQL calls for the rest of the application.

Standalone/JDBC with SQL

The Standalone/JDBC programming model is basically the same as the JCA/JDBC programming model because it uses JDBC with SQL. The difference is that is intended to be used in non-managed environments, which means that you must keep track of your two-phase commit in the XA driver yourself because there is no higher management instance managing it. You also cannot easily predefine standard values in a common way for all of your applications like you can with the JCA drivers. However the syntax and supported SQL functions are the same, making it easily expandable to the JCA approach if necessary.

Standalone/DL/I

The Standalone/DL/I programming model enables the writing of applications with DL/I syntax in standalone Java applications. It uses a DL/I programming API that also enables you to program in the same way you usually code non-Java IMS applications with data access, which gives you the ability to use all IMS database functions in Java if necessary.

Roll-Your-Own/DRDA

The Roll-Your-Own (RYO) approach can be used for all cases where you have the requirement to write non-Java applications that have to access IMS databases through TCP/IP directly. The IMS Open Database feature uses DRDA under the covers for communication. So you must write your own connection socket handling and manually implement all features that you want to have. This is not the recommended programming approach because it is the most complex one. Consider if there are other options for your

requirements, such as a wrapper or middle-layer approach, which uses one of the other programming models.

For more information about the possible DRDA commands, see *IMS Version 11 Application Programming APIs*, SC19-2429.

5.2.4 Comparing the IMS Universal drivers

Depending on your IT infrastructure, solution architecture, and application design, choose the IMS Universal drivers programming approach that is best for your development scenario.

Table 5-3 lists the recommended IMS Universal drivers programming approach to use based on the application programmer's choice of application platform, data access method, and transaction processing option.

Table 5-3 Comparison of IMS Universal driver approaches

Target runtime	Programming approach	Transactionality support	Recommended IMS Universal driver
WebSphere Application Server	JCA/JDBC with SQL	XA and single phase commit with RRS support	imsudbJXA.rar
		Local support only	imsudbJLocal.rar
	CCI with SQL or DL/I	XA and Local-like support	imsudbXA.rar
		Local support only	imsudbLocal.rar
Standalone Java Application	JDBC with SQL	XA (self managed) and local support	imsudb.jar
	DL/I API with DL/I	XA (self managed) and local support	imsudb.jar
Non-Java Application	DRDA with DDM	XA (self implemented) possible	-

Restriction: The XA support is only available with type-4 connectivity.

Note: The IMS Universal Database resource adapters normally have the file extension .rar for Resource Adapter Archive. The IMS Universal Standalone drivers are Java jar files.

The IMS Universal DB resource adapters with XA support have XA support and local-like transaction support. The transaction settings of the application determine which one is used, whether you use container-managed transactions or bean-managed transactions.

The IMS Universal JDBC and the IMS Universal DL/I driver for the Standalone Java applications have XA support and local support implemented. Because there is no higher instance that controls the 2 Phase Commit execution, you must implement them in your application yourself if you want to use the XA support.

For Non-Java applications you can create your own implementations by using the DRDA standard and sending DDM commands to IMS Connect. The application programmer is responsible for implementing the two-phase commit procedure for XA support.

5.3 IMS Universal Database resource adapter

The IMS Universal DB resource adapter is intended to be used in managed Java Enterprise Edition Application Servers (JEE Servers). It makes use of the Java Connector Architecture (JCA) to integrate easily into standard conforming servers. JCA offers many advantages for the programmer because it provides globally-managed services, such as security management, connection pooling, and transaction management, through system contracts without additional coding by the application programmer.

JCA is a Java-based technology solution for connecting application servers and enterprise information systems (EIS) as part of enterprise application integration (EAI) solutions. While JDBC is specifically used to connect Java EE applications to databases, JCA is a more generic architecture for connection to nonstandard systems (including databases). JCA offers a standard interface between the JEE application server and any EIS using a JCA resource adapter.

A resource adapter is a JEE component that implements the JEE Connector architecture for a specific EIS. It is through a resource adapter that a JEE application communicates with the EIS itself. There are two types of contracts (interfaces) implemented by a resource adapter, the application contract and the system contract. The application contract defines the API through which a JEE component, such as an Enterprise bean, accesses the EIS. This API is the only view an application has of the EIS. The system contracts and the resource adapter implementation are transparent to the application component. The EIS specific communication is handled by the resource adapter implementation itself. The system contracts define the interfaces that link the resource adapter to the services that are managed by the JEE server itself. These services include connection, transaction, and security services.

The IMS Universal DB resource adapters with XA support are intended to be used in two-phase commit transaction processing, but you can also use them in single-phase commit applications. The IMS Universal DB resource adapters with local transaction only support provide only single-phase commit functionality.

With the IMS Universal DB resource adapters, you can group interactions in applications together to make sure that all interactions are committed or rolled back. You can do this grouping using container-managed or bean-managed transaction demarcation.

In container-managed transactions, all work that is performed in an EJB method invocation is part of one unit of work, and no explicit demarcation by the application is required. Transactional integrity is managed by the Java EE application server.

Use a bean-managed EJB if you need to have multiple units of works within the same EJB method invocation. In bean-managed transactions, you must use the javax.resource.cci.LocalTransaction or javax.transaction.UserTransaction interface to programmatically demarcate units of work explicitly:

- If you use the LocalTransaction interface, you can only group work that is performed through the used resource adapter.
- Using the UserTransaction interface allows all transactional resources within the application to be grouped.

With the IMS Universal Database resource adapters, you have the choice of two opposite programming styles: the Common Client Interface style and the JCA/JDBC style

Table 5-4 gives a brief overview of the major differences.

Table 5-4 Comparison of JCA Models - CCI and JDBC

Attribute / Function	JDBC	CCI
Data Access Language	SQL	SQL and DL/I
Main Java classes	java.sql	javax.resource.cci
Connection creation	DataSource or driverManager	ConnectionFactory with ConnectionSpec
Command creation	Connection	Connection
Execution commands	Statement	Interaction with SQLInteractionSpec or DLIInteractionSpec
Result	ResultSet	RecordFactory and Records or ResultSet

5.3.1 JCA/Common Client Interface approach

The JCA specification defines the Common Client Interface programming interface. This interface communicates with any Enterprise Information System. The IMS Universal DB resource adapter implements the CCI for interactions with IMS databases. The CCI interfaces for the IMS Universal DB resource adapter are in the com.ibm.ims.db.cci package. The CCI implementation that is provided by IMS allows applications to make SQL and DL/I calls to access the IMS database.

The SQLInteractionSpec class can be used for issuing simple SQL SELECT calls against IMS DBs, and the DLIInteractionSpec class can be used to issue simple DL/I calls against IMS DBs, including RETRIEVE, UPDATE, DELETE, or CREATE.

The following two drivers are available for the IMS Universal DB resource adapter for CCI:

- imsudbXA.rar with XA and local-like transaction support
- imsudbLocal.rar with local transaction support only

Restriction: XA transaction support is only available with type-4 connectivity.

Note: Local transaction support with the XA driver means Single Phase Commit when possible, but it still requires RRS.

Example of a CCI application with SQL calls

Example 5-2 shows an EJB using the CCI programming model. It gets a connection using the JNDI to lookup the Connection Factory in the JEE Server. It also uses SQL with the SQLInteractionSpec class to get a result from the Car Dealer IVP Database.

Example 5-2 CCI with SQL calls

```
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
```

```
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.SQLInteractionSpec;

public class BeanManagedCCISQLBean implements javax.ejb.SessionBean{
   private javax.ejb.SessionContext mySessionCtx;
   public void execute() throws Exception {
      InitialContext ic = new InitialContext();
      ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
      Connection conn = null;
      UserTransaction ut = null;
      try {
         ut = this.mySessionCtx.getUserTransaction();
         ut.begin();
         conn = cf.getConnection();
         Interaction ix = conn.createInteraction();
         SQLInteractionSpec iSpec = new SQLInteractionSpec();
         iSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER WHERE ZIP='12345-6789'");
         ResultSet rs = (ResultSet) ix.execute(iSpec, null);
         while (rs.next()) {
            System.out.println(rs.getString("DLRNAME"));
         }
         rs.close();
         ix.close();
         ut.commit();
         conn.close();
      } catch (ResourceException e) {
         ut.rollback();
         conn.close();
      } catch (SQLException e) {
         ut.rollback();
         conn.close();
      }
   }
   public javax.ejb.SessionContext getSessionContext() {
   return mySessionCtx;
   }
   public void setSessionContext(javax.ejb.SessionContext ctx) {
   mySessionCtx = ctx;
   }
   public void ejbCreate() throws javax.ejb.CreateException {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}
}
```

Example for CCI application with DL/I calls

The coding in Example 5-3 shows you an EJB using the CCI programming model. It gets a connection using JNDI to lookup the Connection Factory in the JEE Server. It uses DL/I calls with the DLIInteractionSpec class to get a result from the Car Dealer IVP Database.

Example 5-3 CCI with DL/I calls

```
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
```

```
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.MappedRecord;
import javax.resource.cci.RecordFactory;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.DLIInteractionSpec;

public class BeanManagedCCIDLIBean implements javax.ejb.SessionBean{
   private javax.ejb.SessionContext mySessionCtx;
   public void execute() throws Exception {
      InitialContext ic = new InitialContext();
      ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
      Connection conn = null;
      UserTransaction ut = null;
      try {
         ut = this.mySessionCtx.getUserTransaction();
         ut.begin();
         conn = cf.getConnection();
         Interaction ix = conn.createInteraction();
         DLIInteractionSpec iSpec = new DLIInteractionSpec();
         iSpec.setFunctionName("RETRIEVE");
         iSpec.setPCBName("AUTOLPCB");
         iSpec.setSSAList("DEALER (DLRNO = '1235')");
         RecordFactory rf = cf.getRecordFactory();
         MappedRecord input = rf.createMappedRecord("DEALER");
         input.put("DLRNAME", null);
         input.put("ZIP", null);
         ResultSet results = (ResultSet) ix.execute(iSpec, input);
         while (results.next()) {
            System.out.println(results.getString("DLRNAME"));
            System.out.println(results.getString("ZIP"));
         }
         results.close();
         ix.close();
         ut.commit();
         conn.close();
      } catch (ResourceException e) {
         ut.rollback();
         conn.close();
      } catch (SQLException e) {
         ut.rollback();
         conn.close();
      }
   }
   public javax.ejb.SessionContext getSessionContext() {
      return mySessionCtx;
   }
   public void setSessionContext(javax.ejb.SessionContext ctx) {
      mySessionCtx = ctx;
   }
   public void ejbCreate() throws javax.ejb.CreateException {}
   public void ejbActivate() {}
   public void ejbPassivate() {}
   public void ejbRemove() {}
}
```

5.3.2 JCA/JDBC approach

In addition to the CCI interface that is provided by the IMS Universal DB resource adapter, you can also write JDBC applications to access your IMS data from a managed environment, while leveraging the Java EE services provided by the application server. This capability is provided by the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter. The IMS Universal JCA/JDBC driver is based on the Java Platform, Enterprise Edition Connector Architecture (JCA) 1.5 and the Java Database Connectivity 3.0 standard.

The following two drivers are available for the IMS Universal JCA adapter with JDBC:

- imsudbJXA.rar with XA and local-like transaction support
- imsudbJLocal.rar with local transaction support only

Restriction: XA transaction support is only available with type-4 connectivity.

Note: Local transactions with the XA driver means Single Phase Commit when possible, but it still requires RRS.

Example for JCA/JDBC application with SQL calls

Example 5-4 shows you an EJB using the JDBC programming model for the IMS Universal DB resource adapter with JDBC. It gets a DataSource Object using the JNDI to lookup the Data Source in the JEE Server. It uses SQL calls through the JDBC API to get a result from the Car Dealer IVP Database.

Example 5-4 JCA/JDBC with SQL calls

```
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.transaction.UserTransaction;

public class BeanManagedJDBCSQLBean {
   private javax.ejb.SessionContext mySessionCtx;
   public void execute() throws Exception {
      InitialContext ic = new InitialContext();
      DataSource ds = (DataSource) ic.lookup("java:comp/env/MyMCF");
      Connection conn = null;
      UserTransaction ut = null;
      try {
         ut = this.mySessionCtx.getUserTransaction();
         ut.begin();
         conn = ds.getConnection();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery("SELECT DLRNAME, ZIP FROM AUTOLPCB.DEALER");
         while (rs.next()) {
            System.out.println(rs.getString("DLRNAME"));
            System.out.println(rs.getString("ZIP"));
         }
         rs.close();
         ut.commit();
         conn.close();
      } catch (SQLException e) {
         e.printStackTrace();
```

```
            ut.rollback();
            conn.close();
        }
    }
        public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
        }
        public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
        }
        public void ejbCreate() throws javax.ejb.CreateException {}
        public void ejbActivate() {}
        public void ejbPassivate() {}
        public void ejbRemove() {}
}
```

5.4 IMS Universal JDBC driver (stand-alone)

The IMS Universal JDBC driver is intended to be used in stand-alone Java applications on the client side. It provides the capability of issuing SQL calls to IMS databases. The IMS Universal JDBC driver is based on the JDBC 3.0 standard.

JDBC is an API that Java applications use to access relational databases or tabular data sources. The JDBC API is the industry standard for database-independent connectivity between the Java programming language and any database that implemented the JDBC interface. The client uses the interface to query and update data in a database. It is the responsibility of the JDBC driver itself to implement the underlying (specific) access protocol for the specific database that the driver is implemented for. Drivers convert requests from Java programs to a protocol that the DBMS can understand.

Using IMS support for JDBC you can write Java applications that can issue dynamic SQL calls to access IMS data and process the result set that is returned in tabular format. The IMS Universal JDBC driver is designed to support a subset of the SQL syntax with functionality that is limited to what the IMS database management system can process natively. Its DBMS-centric design allows the IMS Universal JDBC driver to fully take advantage of the high performance capabilities of IMS. The IMS Universal JDBC driver also provides aggregate function support, ORDER BY support, and GROUP BY support.

There are three ways to establish a connection to IMS with the IMS Universal JDBC driver, using the:

- JDBC DataSource Interface and providing the necessary values directly
- JDBC DataSource Interface and using JNDI to lookup the Connection
- JDBC DriverManager Interface

5.4.1 Connecting to an IMS database using the JDBC DataSource interface

JDBC versions that start with version 2.0 provide the DataSource interface for connecting to a data source. Using the DataSource interface is the preferred way to connect to IMS from your IMS Universal JDBC driver application. This model has the advantage that it allows you to keep your application dynamic using the JNDI to look up the Connection or specifying it manually using the setter methods that belong to the specific DataSource.

Application-managed connection parameters

You can provide all necessary information to access the IMS database directly in your application. The code in Example 5-5 shows how to do this using the JDBC DataSource Interface.

Example 5-5 JDBC DataSource Connection with Application Managed approach

```
import java.sql.*;
import com.ibm.ims.jdbc.*;

public class DataSourceAppManagedApp {
   public static void main(String[] args) {
      Connection conn = null;
      // Create an instance of DataSource
      IMSDataSource ds = new com.ibm.ims.jdbc.IMSDataSource();
      // Set the URL of the fully qualified name of the Java metadata class
      ds.setmetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
      // Set the data store name
      ds.setDatastoreName("IMS2");
      // Set the data store server
      ds.setDatastoreServer("myhost.itso.ibm.com");
      // Set the port number
      ds.setPortNumber(5555);
      // Set the JDBC connectivity driver type
      ds.setdriverType(IMSDataSource.DRIVER_TYPE_4);
      // Disable SSL for connection
      ds.setSSLConnection(false);
      // Set timeout for connection
      ds.setLoginTimeout(10);
      // Set user ID for connection
      ds.setUser("IMSUSR");
      // Set password for connection
      ds.setPassword("myPW");
      // Create JDBC connection
      try {
         conn = ds.getConnection();
         conn.close();
      } catch (SQLException e) {
         e.printStackTrace();
      }
   }
}
```

JNDI-managed connection parameters

The better alternative is to specify the values one time in the Java Naming and Directory Interface and create a connection through obtaining the object from there. You might still override values, such as username, password, or the metadata classes. Through specifying the MetadataURL in your application, you can define only one JNDI connection per IMS and define the used database metadata in your application. The code in Example 5-6 shows you how to do this using the JDBC DataSource Interface and JNDI in a managed environment.

Example 5-6 JDBC DataSource Connection with JNDI Managed approach

```
import java.sql.*;
import com.ibm.ims.jdbc.*;
import javax.naming.*;

public class DataSourceJNDIManagedApp {
   public static void main(String[] args) {
```

```
      try {
         Context ctx = new InitialContext();
         IMSDataSource ds = (IMSDataSource)ctx.lookup("jdbc/imsopendb");
         // Optional Overrides
         ds.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
         ds.setUser("IMSUSR");
         ds.setPassword("myPW");
         // Create Connection
         Connection conn = ds.getConnection();
         conn.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

5.4.2 Connecting to an IMS database using the JDBC DriverManager interface

A JDBC application can also establish a connection to a data source using the JDBC DriverManager interface, as shown in Example 5-7. The JDBC DriverManager interface is part of the java.sql package.

Example 5-7 Connecting with the JDBC DriverManager Interface

```
import java.sql.*;
import java.util.Properties;

public class DriverManagerJDBCApp {
   public static void main(String[] args) {
      try {
         Connection conn = null;
         // Create Properties object
         Properties props = new Properties();
         // Disable SSL for connection
         props.put("sslConnection", "false");
         // Set driverstoreName for connection
         props.put( "driverstoreName", "IMS2" );
         // Set timeout for connection
         props.put("loginTimeout", "10");
         // Set user ID for connection
         props.put( "user", "IMSUSR" );
         // Set password for connection
         props.put( "password", "myPW" );
         // Set URL for the data source
         Class.forName("com.ibm.ims.jdbc.IMSDriver");
         // Create connection
         conn = DriverManager.getConnection("jdbc:ims://myhost.ibm.com:5555/” +
         ”class://samples.ims.openDb.AUTPSB11DatabaseView",props);
         // Close Connection
         conn.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Instead of using the Java.Util.Properties, you can provide the properties directly in the DriverManager.getConnection(url) function. Here is an example of this approach:

```
conn = DriverManager.getConnection("jdbc:ims://myhost.ibm.com:5555/" +
"class://samples.ims.openDb.AUTPSB11DatabaseView:datastoreName=IMS2;" +
"loginTimeout=10;sslConnection=false;user=IMSUSR;password=myPW;");
```

In this case, the necessary values are specified directly in the DriverManagerURL. The disadvantage of using this approach is that your application defines a static connection. When you change, for example, the IP Address or Port, you must recompile your application. The DriverManager Interface is used very often in Tools to create a Connection. The syntax of the DriverManagerURL is:

```
jdbc:ims://<IP>:<PORT>/class://<Package.PSBDatabaseView>:<Parameter1>=<Value1>;
<Parameter2>=<Value2>;
```

5.5 IMS Universal DL/I driver

Use the IMS Universal DL/I driver when you need to write granular queries to access IMS databases directly from a Java client in a non-managed environment.

IMS databases are hierarchical in nature rather than relational; therefore, the JDBC API cannot give you the full capability of all IMS functions. So sometimes it can be useful to use the traditional way of accessing IMS resources, which is through DL/I. The IMS Universal DL/I driver is closely related to the traditional IMS DL/I database call interface that is used with other programming languages for writing applications in IMS.

By using the IMS Universal DL/I driver, you can build segment search arguments (SSAs) and use the methods of the program communication block (PCB) object to read, insert, update, delete, or perform batch operations on segments. You can gain full navigation control in the segment hierarchy.

5.5.1 Basic steps in writing an IMS Universal DL/I driver application

If you are familiar with DL/I processing, the usage of the DL/I API is very similar. It gives you the ability to work with the traditional programming model in a new environment, such as Java. This book concentrates on leveraging industry standards that in principle are JDBC and SQL programming. To explain DL/I in detail is too much for the purpose of this book. Therefore we mention briefly the general steps of writing an application program with the IMS Universal DL/I driver:

1. Import the com.ibm.ims.dli and com.ibm.ims.base packages that contain the IMS Universal DL/I driver classes, interfaces, methods, and exceptions.
2. Create an IMSConnectionSpec instance by calling the create IMSConnectionSpec method in the IMSConnectionSpecFactory class.
3. Set the connection properties for the IMSConnectionSpec instance.
4. Obtain a program specification block (PSB), which contains one or more PCBs by passing the connection request properties to the PSBFactory class to create the PSB instance. When the PSB instance is created successfully, a connection is established to the database.
5. Obtain a PCB handle, which defines an application's view of an IMS database and provides the ability to issue database calls to retrieve, insert, update, and delete database information.

6. Obtain an unqualified segment search argument list (SSAList) of one or more segments in the database hierarchy.
7. Add qualification statements to specify the segments targeted by DL/I calls.
8. If retrieving data, mark the segment fields to be returned.
9. Execute DL/I calls to the IMS database.
10. Handle errors that are returned from the DL/I programming interface.
11. Disconnect from the IMS database subsystem.

For more Information about DL/I programming with the IMS Universal DL/I driver see *IMS V11 Application Programming Guide*, SC19-2428 and *IMS Version 11 Application Programming APIs*, SC19-2429.

5.5.2 Sample code using the IMS Universal DL/I driver

For a detailed explanation and code samples using the IMS Universal DL/I driver, see Chapter 8, “Scenario 3: Writing DL/I and mixed applications” on page 181.

5.6 SQL syntax for the IMS Universal drivers

The acronym SQL stands for structured query language and is used to query and manipulate data in relational database management systems. IMS databases are hierarchical in nature. Hierarchical databases have many advantages that enable efficient programming in many areas, including XML and sequential batch processing of data.

The data access language of IMS for the hierarchical databases is traditionally DL/I. Today SQL is the de-facto industry standard for the data access language in applications, which is not a problem for IMS because its data can be accessed relationally and hierarchically. IMS can provide a relational view of your hierarchical IMS Databases and make it accessible using SQL, and you can use traditional programming languages for hierarchical database access in new environments. At the same time, you do not have to change your existing DL/I applications.

The Java database metadata class contains information about the IMS database, including segments, segment names, the segment hierarchy, fields, field types, field names, fields offsets, and field lengths. The metadata is used by the IMS Universal JDBC drivers to allocate program-specification blocks (PSBs), issue DL/I calls, perform data transformation, and translate SQL queries to DL/I calls.

Example 5-5 on page 115 shows the mapping between hierarchical database terms and relational database terms.

Table 5-5 Mapping between IMS terms and relational terms

Hierarchical IMS DB term	Relational DB equivalent
Segment name	Table name
Segment instance	Table row
Segment field name	Column name
Segment unique key	Table primary key
Virtual foreign key field	Table foreign key

Hierarchical IMS DB term	Relational DB equivalent
PCB	Database View
PSB	Collection of Database Views

In the following sections, we give an overview of the most important SQL statements and how to use them with the IMS Universal drivers supporting JDBC.

5.6.1 SQL keywords

If you use a SQL keyword as a name for a PCB, segment, or field, your JDBC application program returns an error when it attempts an SQL query. The keywords are not case-sensitive. Table 5-6 lists the SQL keywords.

Table 5-6 SQL keywords

ALL	DISTINCT	SELECT
AND	FROM	SET
AS	GROUP BY	SUM
ASC	INSERT	UPDATE
AVG	MAX	VALUES
COUNT	MIN	WHERE
DELETE	OR	
DESC	ORDER BY	

In addition to the IMS-supported SQL keywords listed in Table 5-6, there are several more keywords that are known by the SQL syntax. These keywords are restricted by the IMS Universal drivers because they might be used in future releases of the IMS Universal drivers. Therefore, we recommend that you not use these keywords as column names.

Note: The IMS Enterprise Suite DLIModel Utility detects if these keywords are used as a field or segment name and renames them. In our Car Dealer Database, for example, the case for the COUNT field in the MODEL Segment is renamed to COUNT1.

Table 5-7 on page 120 lists the restricted SQL keywords.

Table 5-7 Restricted SQL keywords

ABORT	CROSS	IS	REAL
ANALYZE	CURRENT	JOIN	REFERENCES
AND	CURSOR	LAST	RESET
ALL	DECIMAL	LEADING	REVOKE
ALLOCATE	DECLARE	LEFT	RIGHT
ALTER	DEFAULT	LIKE	ROLLBACK
AND	DELETE	LISTEN	SELECT
ANY	DESC	LOAD	SET
ARE	DISTINCT	LOCAL	SETOF
AS	DO	LOCK	SHOW
ASC	DOUBLE	MAX	SMALLINT
ASSERTION	DROP	MIN	SUBSTRING
AT	END	MOVE	SUM
AVG	EXECUTE	NAMES	TABLE
BEGIN	EXISTS	NATIONAL	TO
BETWEEN	EXPLAIN	NATURAL	TRAILING
BINARY	EXTRACT	NCHAR	TRANSACTION
BIT	EXTEND	NEW	TRIM
BOOLEAN	FALSE	NO	TRUE
BOTH	FIRST	NONE	UNION
BY	FLOAT	NOT	UNIQUE
CASCADE	FOR	NOTIFY	UNLISTEN
CAST	FOREIGN	NULL	UNTIL
CHAR	FROM	NUMERIC	UPDATE
CHARACTER	FULL	ON	USER
CHECK	GRANT	OR	USING
CLOSE	GROUP	ORDER	VACUUM
CLUSTER	HAVING	OUTER	VALUES
COLLATE	IN	PARTIAL	VARCHAR
COLUMN	INNER	POSITION	VARYING
COMMIT	INSERT	PRECISION	VERBOSE
CONSTRAINT	INT	PRIMARY	VIEW
COPY	INTEGER	PRIVILEGES	WHERE
COUNT	INTERVAL	PROCEDURE	WITH
CREATE	INTO	PUBLIC	WORK

5.6.2 Primary key and virtual foreign key handling

In relational databases, the relationships are built using foreign key relationships between tables. In IMS, the relationships are part of the hierarchy itself. The IMS Universal JDBC driver introduces the concept of virtual foreign keys to capture these explicit hierarchies in a relational sense.

Figure 5-2 on page 121 shows an extract of the Car Dealer IVP Database Diagram created by the IMS Enterprise Suite DLIModel utiltiy using the AUTOLPCB.

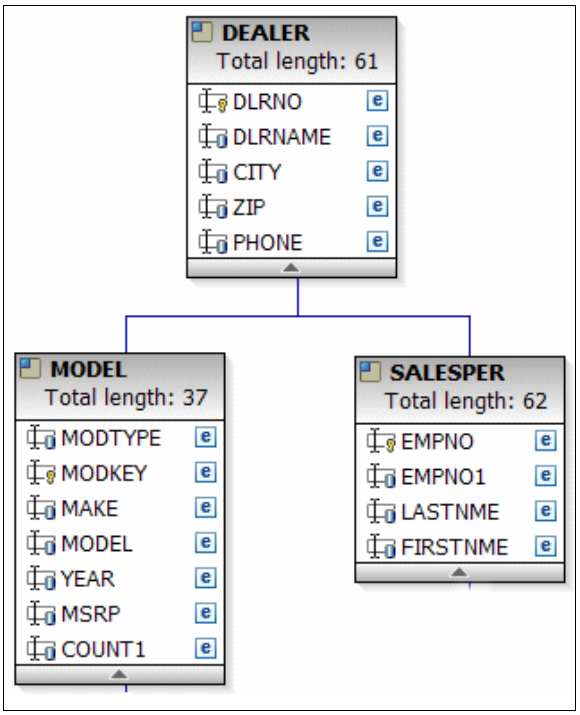


Figure 5-2 Extract from overview diagram from Car Dealer IVP Example

The DEALER segment is the root segment in this view and has two child segments MODEL and SALESPER. The DLRNO field is the primary key of the DEALER segment. MODKEY is the primary key of the MODEL segment.

Every table that is not the root table in a hierarchic path will virtually contain the unique keys of all of its parent segments up to the root of the database. These keys are called virtual foreign key fields.

The IMS Universal JDBC drivers automatically generate a Virtual Foreign Key Column based on the Segment Names. The following SELECT query shows an example.

```
SELECT * FROM AUTOLPCB.MODEL
```

This statement returns all columns from the MODEL segment that are physically stored in the database and one more column called DEALER_DLRNO that contains the reference to the parent segment's primary key field. This field is not stored physically in the database. Figure 5-3 shows the result of this query.

DEALER_DLRNO	MODTYPE	MODKEY	MAKE	MODEL	YEAR	MSRP	COUNT1
1234	S	FORD FO...	FORD	FOCUS	2002	17995	03
1235	S	Volvo S40 ...	Volvo	S40	2002	21000	01
1236	S	TRABANT S...	TRABANT	SUPERIOR	1988		03

Figure 5-3 Query result from MODEL table

The purpose of the virtual foreign key fields is to maintain referential integrity, similar to foreign keys in relational databases, which allows SQL SELECT, INSERT, UPDATE, and DELETE queries to be written against specific tables and columns located in a hierarchic path.

5.6.3 Using the SELECT statement

Use the SELECT statement to retrieve data from one or more tables. The result is returned in a tabular result set.

When using the SELECT statement with the IMS Universal JDBC driver:

- If you are selecting from multiple tables and the same column name exists in one or more of these tables, you must table-qualify the column or an ambiguity error will occur.
- The FROM clause must list all the tables that you are selecting data from. The tables listed in the FROM clause must be in the same hierarchic path in the IMS database.
- In Java applications that use the IMS JDBC drivers, connections are made to PSBs. Because there can be multiple database PCBs in a PSB, queries must specify which PCB in a PSB to use. To specify which PCB to use, always qualify segments that are referenced in the FROM clause of an SQL statement by prefixing the segment name with the PCB name. You can omit the PCB name only if the PSB contains only one PCB.

Here are several examples of valid SELECT statements against the Car Dealer IVP Database:

This statement returns all rows and all columns from the DEALER segment using the AUTOLPCB:

```
SELECT * FROM AUTOLPCB.DEALER
```

This statement returns the DLRNAME column from all rows where the PHONE equals the search string from the DEALER segment in the AUTOLPCB:

```
SELECT DLRNAME FROM AUTOLPCB.DEALER WHERE PHONE=6667777
```

This statement returns the MODEL column from all distinct rows from the MODEL segment using the AUTOLPCB:

```
SELECT DISTINCT MODEL FROM AUTOLPCB.MODEL
```

This statement returns all rows from the ZIP, CITY, and DLRNAME columns. The DLRNAME column is referenced as NAME. The Resultset is ordered by ZIP ascendingly by default:

```
SELECT ZIP,CITY,DLRNAME AS NAME FROM AUTOLPCB.DEALER ORDER BY ZIP
```

This statement returns all rows from the columns MAKE and MODEL grouped by MAKE (AS BRAND) and MODEL. It is also ordered by the column MAKE:

```
SELECT MAKE AS BRAND,MODEL FROM AUTOLPCB.MODEL GROUP BY BRAND,MODEL ORDER BY
MAKE
```

This statement returns the highest YEAR from the MODEL column. The column name is a combination of the aggregate function name and the field name separated by an underscore character (_). In this case, you get the result using resultSet.getInt("MAX_YEAR"). If the aggregate function argument field is table-qualified, the ResultSet column name is the combination of the aggregate function name, the table name, and the column name, separated by underscore characters (_), for example, SELECT MAX(Model.year) results in a column name MAX_Model_year:

```
SELECT MAX(YEAR) FROM AUTOLPCB.MODEL
```

This select statement returns data from the DEALER root segment as well as from the MODEL segment. It uses the virtual foreign key DEALER_DLRNO field in the DEALER table. So it will give you all entries from the MODEL database and the referred parent entries in the DEALER table:

```
SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE MODEL.DEALER_DLRNO =
DEALER.DLRNO
```

5.6.4 Using the INSERT statement

The INSERT statement inserts new rows into a table.

The following sample SQL shows the insert of a record at root level:

```
INSERT INTO AUTOLPCB.DEALER (DLRNO, ZIP, DLRNAME, CITY, PHONE)  VALUES ('8888',
'71139', 'Thilo', 'Stuttgart', '555-888')
```

When inserting a record in a table at a non-root level, you must specify values for all of the virtual foreign key fields of the table:

```
INSERT INTO AUTOLPCB.MODEL (DEALER_DLRNO, MODTYPE, MAKE, MODEL,  YEAR, MSRP,
COUNT1)  VALUES ('8888', 'S', 'LIDLA', 'SuperABC', '2010', '66000', '05')
```

This insert statement creates a MODEL record which refers to the DEALER segment with the primary key DLRNO=8888.

Attention: The MODKEY field must not be inserted because it is a primary key field and is automatically built by combining the MAKE, MODEL, and YEAR values. Alternately, you can insert the MODKEY in the proper format and therefore do not specify the MAKE, MODEL, and YEAR.

5.6.5 Using the UPDATE statement

The UPDATE statement is used to update existing records in a table.

The following sample SQL shows an update in a dependent table:

```
UPDATE AUTOLPCB.MODEL SET MSRP='50000' WHERE DEALER_DLRNO = '8888' AND MSRP >=
'60000'
```

This UPDATE statement sets the MSRP value to 50000 for all MODEL records that refer to DEALER with the DLRNO = 8888 and where the MSRP value is greater than or equal to 60000.

Restriction: Updates to a virtual foreign key field will fail because they do not exist in the database. To change a dependent segment, you must delete the segment and reinsert it referring to the segment you want to put it under.

5.6.6 Using the DELETE statement

The DELETE statement removes rows in a table. DELETE operations are cascaded to all child segments.

The following sample SQL shows the usage of the DELETE statement.

Deleting the root Segment DEALER without a WHERE clause, as in the following statement, deletes the entire database and all its records:

```
DELETE FROM AUTOLPCB.DEALER
```

The following statement deletes all entries in the MODEL table and all referenced segments under it in the database.

```
DELETE FROM AUTOLPCB.MODEL
```

The following SQL statement deletes the root record of DLRNO= 8888 and all its dependent segments.

```
DELETE FROM AUTOLPCB.DEALER WHERE DLRNO =’8888‘
```

The following SQL statement deletes the MODEL segments and all their dependent segments for the DEALER with DLRNO= 8888:

```
DELETE FROM AUTOLPCB.MODEL WHERE DEALER_DLRNO =’8888’
```

5.6.7 Using the WHERE statement

The WHERE statement can be used in combination with the SELECT, UPDATE, and DELETE statements. It specifies an exact record or filter results in a query.

You can use the following operators in the WHERE clause to select data conditionally:

```
=     Equals
!=    Not equal
>     Greater than
>=    Greater than or equals
<     Less than
<=    Less than or equals
```

Note: Our recommendation is to compare columns to values and not to other columns. This type of comparison is better for the performance of the queries and helps to reduce errors in your SQL statements where incorrect results are returned. However, it is legal to compare the virtual foreign key column with another primary key column.

The usage rules for the WHERE statement are:

- Do not use parentheses. Qualification statements are evaluated from left to right. The order of evaluation for operators is the IMS evaluation order for segment search arguments.
- List all qualification statements for a table adjacently, for example, in the following valid WHERE clause, the qualified columns from the same DEALER table are listed adjacently:

  ```
  SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE AUTOLPCB.DEALER.ZIP =
  '88888' OR AUTOLPCB.DEALER.ZIP = '88888' AND AUTOLPCB.MODEL.MODEL = 'B Plus'
  ```

  The following statement is an invalid WHERE clause because they are not grouped together:

  ```
  SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE AUTOLPCB.DEALER.ZIP =
  '88888' AND AUTOLPCB.MODEL.MODEL = 'B Plus' OR AUTOLPCB.DEALER.ZIP = '88888'
  ```

- The OR operator can be used only between qualification statements that contain columns from the same table. To combine qualification statements for separate tables, use an AND operator.
- The columns in the WHERE clause must be DBD-defined fields. The only exception to this is when the columns in the WHERE clause are sub fields that make up a DBD-defined field.

  Note: The columns that are in the DBD are marked in the DLIModel IMS Java report as being either primary key fields or search fields.

  For example, a DBD-defined field is named ADDRESS and is 30 bytes long. In a COBOL copybook, this field is broken down into CITY, STATE, and ZIP sub fields as illustrated by the following code:

```
01 ADDRESS
   02 CITY PIC X(10)
   02 STATE PIC X(10)
   02 ZIP PIC X(10)
```

  Without the sub field support, the ADDRESS value in the WHERE clause must be manually padded and entered like this:

```
WHERE ADDRESS = 'Stuttgart GER       70565     '
```

  With the sub field support, you can enter the WHERE clause like this:

```
WHERE CITY = 'Stuttgart' AND STATE = 'GER' AND ZIP = '70565'
```

  The following rules apply for sub fields:

  – Sub fields must always be fully defined because it is only possible to search for the whole DBD-defined search-field.
  – The columns in the WHERE clauses are not allowed to be separated and must be combined by an AND operator.
  – The EQUAL (=) is the only allowed operator on sub fields.

5.6.8 Using aggregate functions

The following aggregate functions are supported by the IMS Universal drivers:

– AVG
– COUNT
– MAX
– MIN
– SUM

The following supported keywords can also be used to aggregate the results or influence the previously mentioned aggregate functions:

– AS
– DISTINCT
– GROUP BY
– ORDER BY ASC/DESC

The ResultSet column name from an aggregate function is a combination of the aggregate function name and the field name separated by an underscore character (_). The examples in Table 5-8 on page 126 show the SELECT query and the example method of getting the column results from the ResultSet.

Table 5-8 Aggregate functions examples

SELECT statement	ResultSet column name
SELECT MAX(year)	rs.getInt("MAX_year")
SELECT MAX(MODEL.year)	rs.getLong(“MAX_MODEL_year”)
SELECT MAX(MODEL.year) AS ‘oldest’	rs.getInt("oldest")
SELECT COUNT(DEALER.DLRno)	rs.getInt(“COUNT_DEALER_DLRno”)
SELECT COUNT(DISTINCT DEALER.DLRNAME)	rs.getInt(“COUNT_DISTINCT_DEALER_DLRNAME”)
SELECT COUNT(DISTINCT AUTOLPCB.DEALER.DLRNAME)	rs.getInt(“COUNT_DISTINCT_DEALER_DLRNAME”)

Note: The schema name (PCB name) is not added to the column name.

Attention: Only the COUNT aggregate function allows the DISTINCT keyword in it.

Table 5-9 shows the allowed aggregate functions arguments and the corresponding result types.

Table 5-9 Aggregate functions and result types

Aggregate function	Argument type	Result type
SUM and AVG	Byte	Long
	Short	Long
	Integer	Long
	Long	Long
	BigDecimal	BigDecimal
	Single-precision floating point	Double-precision floating point
	Double-precision floating point	Double-precision floating point
MIN and MAX	Any type except BIT, BLOB, or BINARY	Same as argument type
COUNT	Any type	Long

A type conversion can be done by using the ResultSet.getXXX() function as long as it is a valid conversion.

5.7 Data transformation support

Using the IMS Universal drivers you can easily transform data from the format defined in the metadata to another supported Java format. You can do this transformation using the get<type> function of the ResultSet in JDBC applications or using these functions on the path object in DL/I applications.

5.7.1 JDBC data types to Java data types mapping

Table 5-10 shows the mapping between JDBC data types, which are used in the IMS metadata class file to define the data type of the fields in the IMS database and the Java data types.

Table 5-10 JDBC data types to Java data types mapping

JDBC data type	Java data type	Length
BIGINT	long	8 bytes
BINARY	byte[]	1-32 KB
BIT	Boolean	1 byte
CHAR	java.lang.String	1-32 KB
DATE	java.sql.Date	Application-defined
DOUBLE	double	8 bytes
FLOAT	float	4 bytes
INTEGER	int	4 bytes
PACKEDDECIMAL [a]	java.math.BigDecimal	1-10 bytes
SMALLINT	short	2 bytes
TIME	java.sql.Time	Application-defined
TIMESTAMP	java.sql.Timestamp	Application-defined
TINYINT	byte	1 byte
VARCHAR	java.lang.String	1-32 KB
ZONEDECIMAL [a]	java.math.BigDecimal	1-19 bytes

a. Data types of the IMS drivers (not in the JDBC specification)

5.7.2 Compatible data transformation functions

Table 5-11 on page 128 shows the available get methods in the ResultSet or Path Interface for accessing data of a certain JDBC type.

In Table 5-11 on page 128, those marked with:

- **X** indicate methods designed for accessing the given data type. No truncation or data loss will occur using those methods.
- O indicate all other legal calls; however, data integrity cannot be ensured using those methods.
- If the box is empty (it neither contains an X or an O), using that method for a data type will result in an exception.

Table 5-11 Available get methods for data types

JDBC Type	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	DOUBLE	BIT	CHAR	VARCHAR	PACKEDDECIMAL	ZONEDECIMAL	BINARY	DATE	TIME	TIMESTAMP
getByte	**X**	O	O	O	O	O	O	O	O	O	O				
getShort	O	**X**	O	O	O	O	O	O	O	O	O				
getInt	O	O	**X**	O	O	O	O	O	O	O	O				
getLong	O	O	O	**X**	O	O	O	O	O	O	O				
getFloat	O	O	O	O	**X**	O	O	O	O	O	O				
getDouble	O	O	O	O	O	**X**	O	O	O	O	O				
getBoolean	O	O	O	O	O	O	**X**	O	O	O	O				
getString	O	O	O	O	O	O	O	**X**	**X**	O	O	O	O	O	O
getBigDecimal	O	O	O	O	O	O	O	O	O	**X**	**X**				
getBytes												**X**			
getDate									O	O			**X**		O
getTime									O	O				**X**	O
getTimestamp									O	O			O	O	**X**

Note: At this time PackedDecimal and ZoneDecimal data types do not support the Sign Leading or Sign Separate modes. Sign information is always stored with the Sign Trailing method.

Note: PackedDecimal and ZoneDecimal are not data types of the JDBC specification.

6

Scenario 1: JDBC data access through tooling

This chapter shows you the possibilities of the IMS Open Database feature for using your existing programs and tools. What is shown in this chapter works with other products that, as are the IMS Universal DB drivers, are built on open standards, such as JDBC. They should be compatible with all tools supporting that standard as long as the IMS Universal JDBC driver and the metadata class file are available from the application class path. This solution opens access to IMS data for many available products and open source projects.

In this chapter, we demonstrate the following tools and products, which you might already have in your installation or might want to try out for this purpose:

- IBM Data Perspective in Data Studio and Rational products
- Accessing IMS Data in Cognos
- Accessing IMS Data using the IBM Mashup Center

6.1 IBM Data Perspective in Data Studio and Rational products

A Perspective is an arrangement of views in Eclipse. IBM integrated a Data Perspective in several of their Eclipse-based products. In this case, it is an arrangement of useful views concerning database access, maintenance, and application development. It is very much focused on databases with SQL access. With IMS Version 11 and the IMS Universal DB drivers, it now becomes easily usable for IMS database access and application development.

6.1.1 Downloading and installing IBM Data Studio or Rational products

You can use what is shown here with Rational Application Developer, Rational Developer for System z, and IBM Data Studio depending on which product you already have in your company. Besides these products, there are several open source projects and other vendor products that are available with similar capabilities, which might be configurable and usable in a similar way.

If you do not have one of these products, you can try them out by downloading and installing them on your workstation:

- A special Rational Developer for System z Version for IMS can be downloaded from the IBM IMS Enterprise Suite web site:

  `http://www.ibm.com/software/data/ims/soa-integration-suite/enterprise-suite/`

- A trial version of Rational Application Developer:

  `http://www.ibm.com/developerworks/downloads/r/rad/`

- IBM Data Studio (a no-charge product) is available at:

  `http://www.ibm.com/software/data/optim/data-studio/features.html`

In this example, we show the capabilities of the Data Perspective along with the IBM Data Studio. For the no-cost version of IBM Data Studio, there are currently two packages available: the stand-alone package and the integrated development environment (IDE) package.

- The *stand-alone package* is a lighter weight offering that is ideal for administrators to get up and running quickly and easily. This package is sufficient for most users who do standard database administration or database routine development and includes the new query formatting and pureScale™ administration support. The stand-alone package does not include SQLJ, Data Web Services, or XML development capability. If you need that capability, download the integrated development environment package described in the next bullet.

- The *integrated development environment* package includes most administrative capabilities and an integrated Eclipse development environment for Java, XML (for DB2 for Linux, UNIX, and Windows), and Web Services development. You can install this package with compatible Eclipse-based products, such as InfoSphere™ Data Architect, Optim™ Development Studio, Optim Database Administrator, and Optim Query Tuner within the same Eclipse instance (shell share), providing seamless movement between data-centric roles or to share objects across geographically distributed teams.

In our case, we chose the IDE package for download. The following steps briefly describe how to install the product:

1. Go to the IBM Data Studio web site, and select the IDE Package for download.

   You need an IBM ID for the download, which can be requested on the web site (if you do not have one already).

2. Select the correct download for your Workstation Operating System, as shown in Figure 6-1, and click **Continue**.

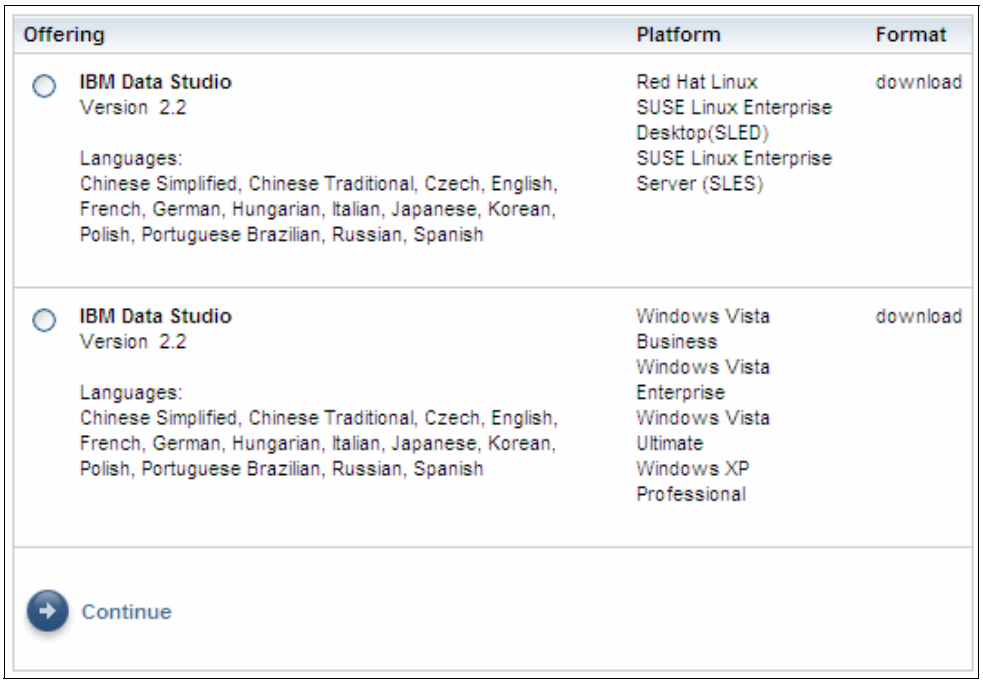


Figure 6-1 Data Studio Installation - Operating System Choice

3. Download the Data Studio Install .zip file.
4. After Downloading the .zip file, extract it to a temporary directory.
5. To install it, you have two options:
   - Using the Launchpad that is part of the installation .zip file. Change to the extracted directory, and start setup.exe.
   - If you already have IBM Installation Manager installed, you can use the Installation Manager, and add the disk1 directory of the extracted files to the Repository of the IBM Installation Manager. This approach is necessary if you want to do shell-sharing with other Eclipse Products or if you want to do all installations with one product.
6. In our case, we chose the first option in the previous step and continued the installation steps on the panel until completion.

If you need help with or more information about the installation, see the IBM Data Studio Information Center at:

http://publib.boulder.ibm.com/infocenter/idm/v2r2/topic/com.ibm.datatools.ds.install.doc/topics/t_install_over.html

6.1.2 Configuring IBM Data Studio for use with the IMS Universal JDBC driver

After installing IBM Data Studio, you must configure it for use with the IMS Universal JDBC driver.

Tip: The following steps are similar in other Eclipse-based products that have the IBM Data Perspective.

To configure IBM Data Studio for use with the IMS Universal JDBC driver:

1. You can start IBM Data Studio from the Windows Start Menu or a Desktop Shortcut. It asks you for a directory for its workspace. The workspace is a folder where all Eclipse-based settings and your projects are stored. If the directory is not there, it is automatically created. Figure 6-2 shows an example of our workspace selection.

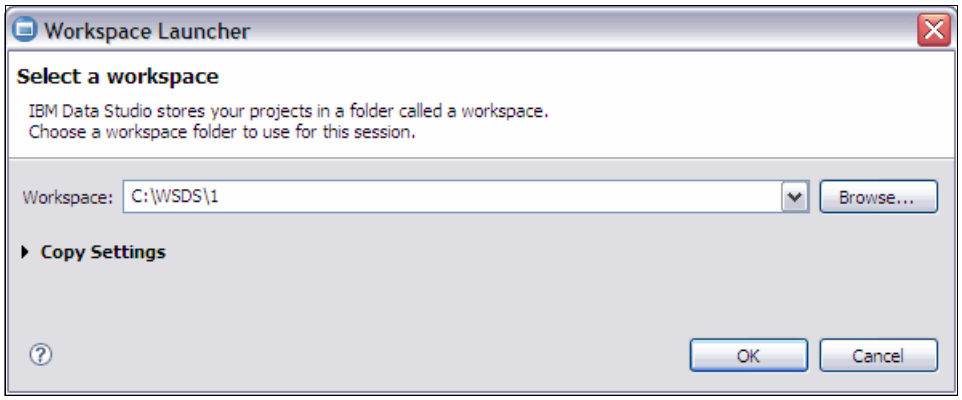


Figure 6-2 Data Studio - Workspace selection

2. When you start IBM Data Studio for the first time or with a new workspace, it shows the Welcome panel that is shown in Figure 6-3.

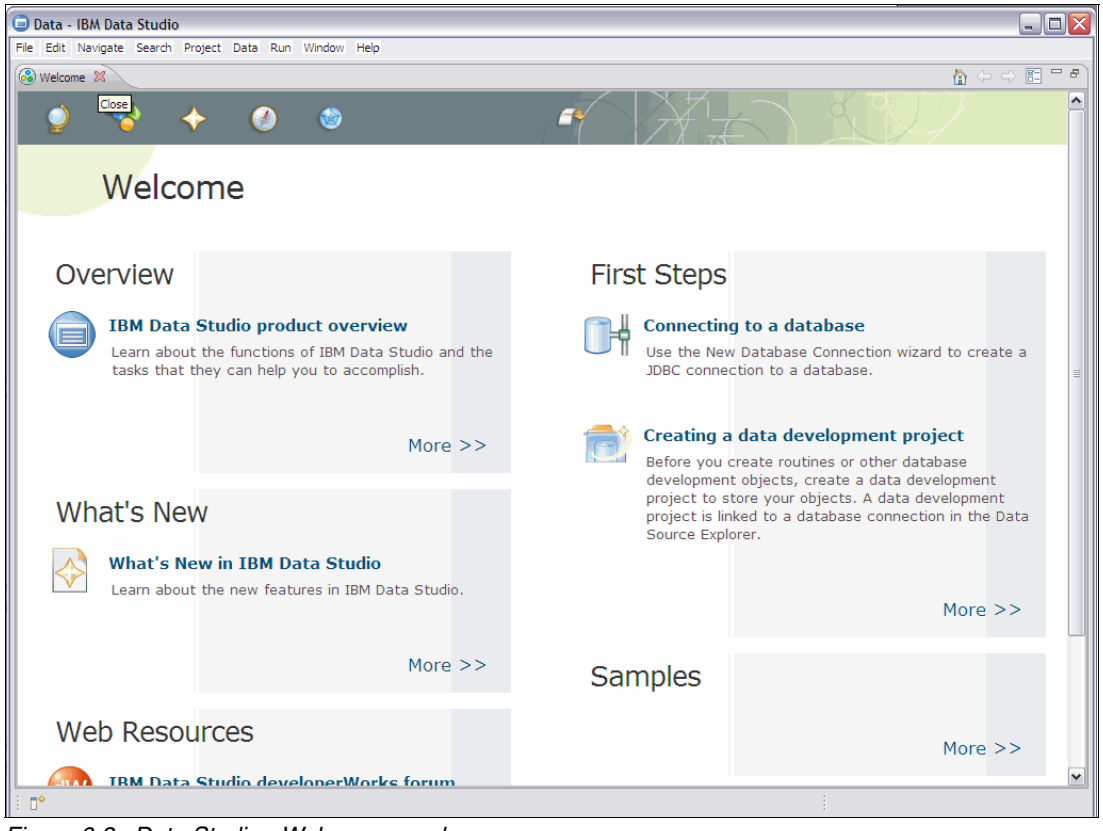


Figure 6-3 Data Studio - Welcome panel

3. To close the Welcome panel, click the X on the top tab bar, and it automatically shows you the Data Perspective.

4. From the menu select **Window** → **Preferences.** In the Menu, drill down to **Data Management** → **Connectivity** → **Driver Definitions**, and select the predefined **Generic JDBC 1.0 Driver**, as shown in Figure 6-4.

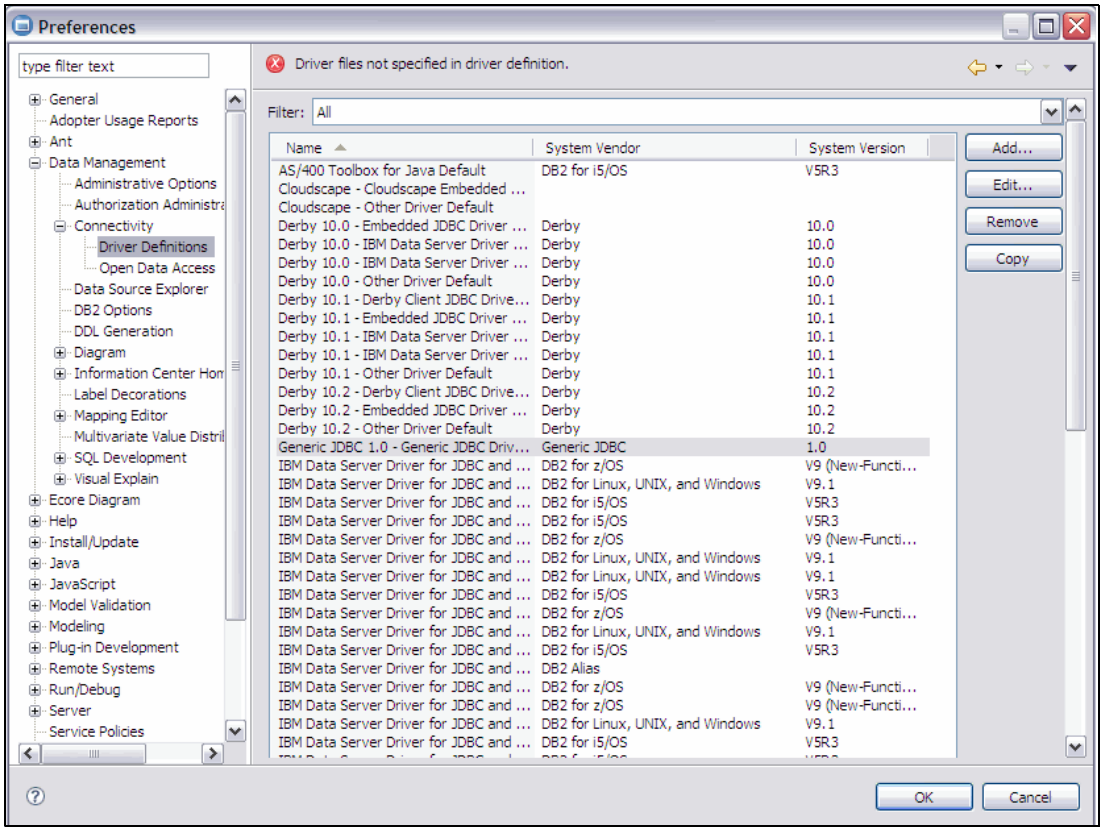


Figure 6-4 Data Studio - Configuring IMS Universal Drivers Step 1

5. Click **Edit**, and switch in the appearing panel to the Jar List tab.
6. Click **Add Jar/Zip**, and select the **imsudb.jar**[1] as the IMS Universal Driver on your hard disk.
7. Repeat the previous step, and add the Database Metadata Jar files you want to access. In our example, this is **AUTPSB11.jar** which is generated as shown in Chapter 4, “Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility” on page 77. In our example, the Jar List tab is shown in Figure 6-5 on page 134.

[1] You previously downloaded the IMS Universal drivers using FTP in binary mode.

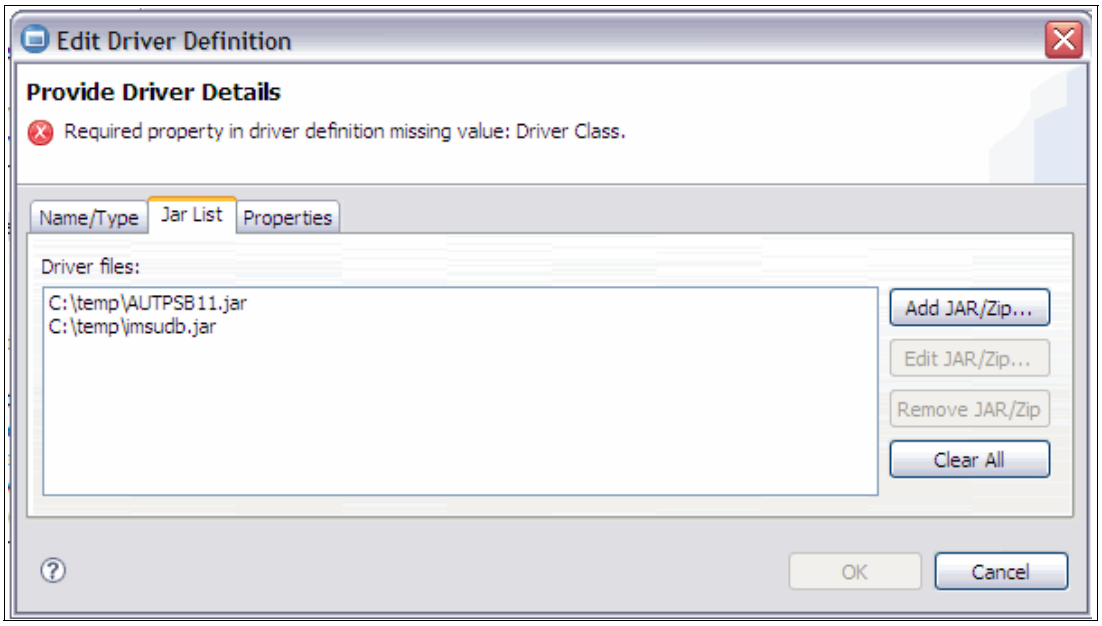


Figure 6-5 Data Studio - Configuring IMS Universal Drivers Step 2

8. Switch to the Properties tab, and click in the Driver Class Value field.
9. Select **Browse for class**, and select **com.ibm.ims.jdbc.IMSDriver**, as shown in Figure 6-6, and click **OK**.

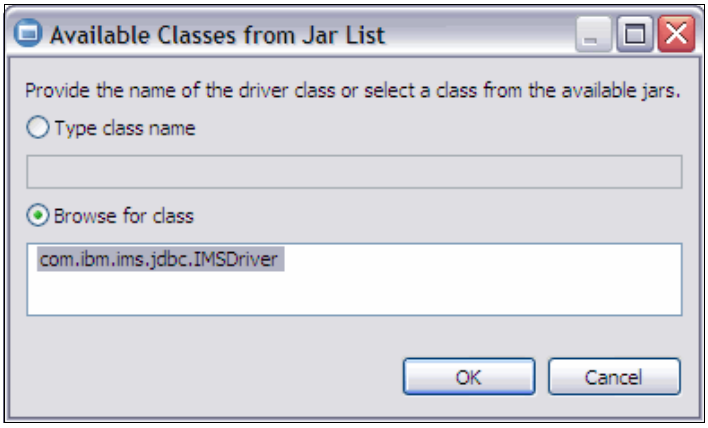


Figure 6-6 Data Studio - Configuring IMS Universal Drivers Step 3

10. You can now optionally specify the Connection URL template, which is then the default for new connections. We set it to **jdbc:ims://host:5555/class://samples.xDatabaseView:datastoreName=IMS2** as well as the Database Name and the Default User ID, as shown in Figure 6-7 on page 135. Click **OK**.

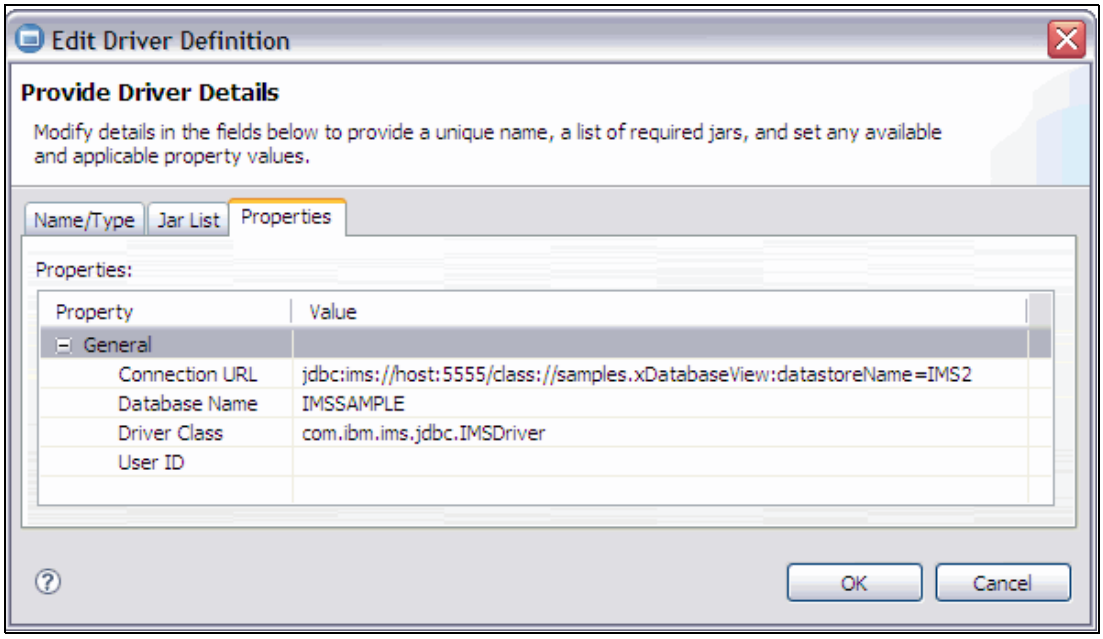


Figure 6-7 Data Studio - Configuring IMS Universal Drivers Step 4

11. Click **Finish**. You completed the set up of the workstation and can now use the Data Perspective in your product.

6.1.3 Using the Data Perspective with the IMS Universal Drivers

This section aims to give you a brief overview of the interesting features of the Data Perspective and how they work with the IMS Universal Drivers.

Creating a connection to IMS

To create a connection to IMS:

1. First, you must create a connection to an IMS Database. In the Data Source Explorer, right-click **Database Connections**, and select **New**, as shown in Figure 6-8.

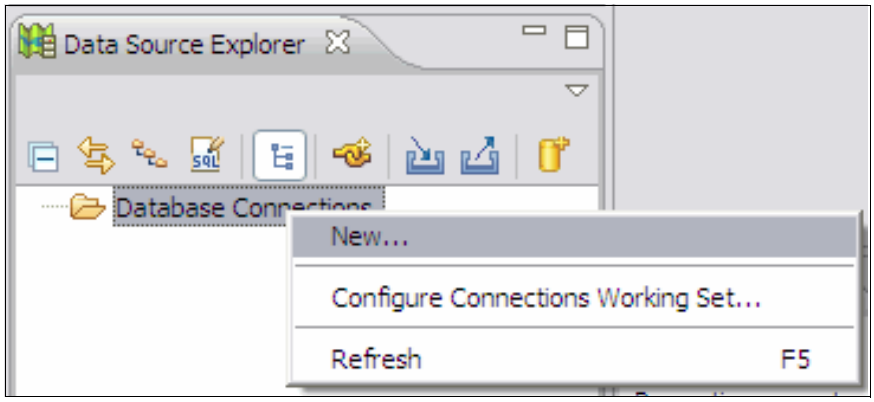


Figure 6-8 Data Studio - Creating a new Connection Step 1

2. Select **Generic JDBC** in the list on the left, and complete the necessary information regarding the correct URL, User name, and Password. In our case, the URL is

jdbc:ims://myhost:5555/class://samples.ims.openDb.AUTPSB11DatabaseView:dataStoreName=IMS2;dpsbOnCommit=true, as shown (truncated) in Figure 6-9.

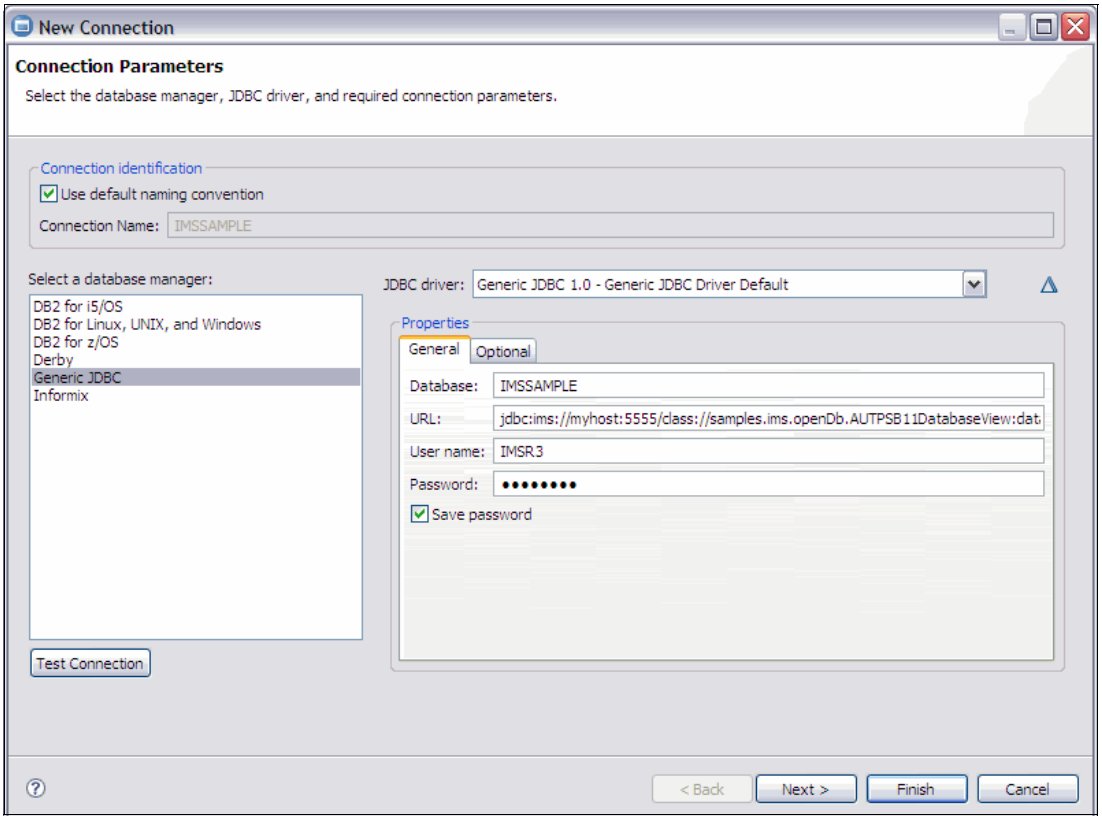


Figure 6-9 Data Studio - Creating a new Connection Step 2

Note: The dpsbOnCommit=true parameter for the URL is recommended in environments that are doing an internal pooling of the connections without notifying the driver.

3. Click **Finish**, and your new connection opens in the Data Source Explorer.

   For details about Data Source Explorer, see:

   http://publib.boulder.ibm.com/infocenter/idm/v2r1/index.jsp?topic=/com.ibm.datatools.server.ui.doc/topics/cdatabaseexplorer.html

Returning sample data through Data Source Explorer

Now that we created the connection, we can use it in the following steps:

1. Drill down and expand the connection until you see the schemas (PCBs), the tables (segments), and columns (fields), as shown in Figure 6-10 on page 137.

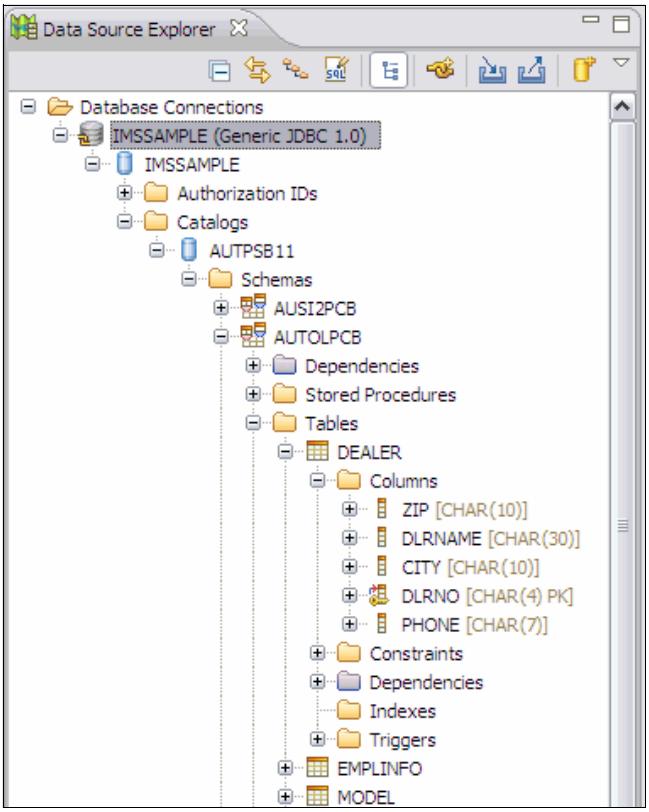


Figure 6-10 Data Studio - AUTPSB11 expanded view

2. Right-click a table, and select **Data** → **Return All Rows**, as shown in Figure 6-11.

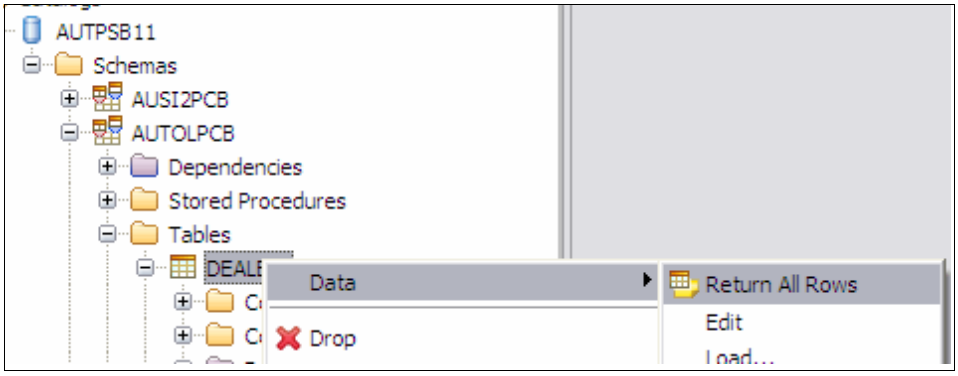


Figure 6-11 Data Studio - Return All Rows

The results received from the IMS database are shown in Figure 6-12 on page 138.

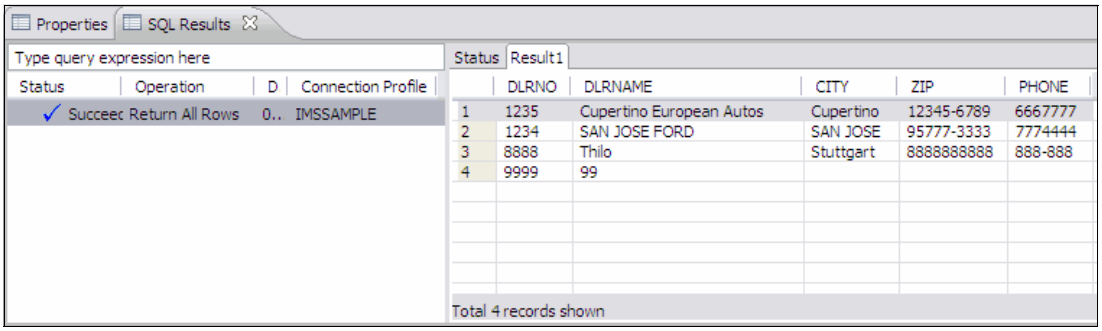


Figure 6-12 Data Studio - Return all Rows - Results

This is a way to easily return sample data from a specific segment in an IMS database.

Editing data using the Data Source Explorer

It is also possible to edit the data in a tabular form with the Data Source Explorer:

1. Right-click a table (for example, AUTOLPCB.SALESINF) in the Data Source Explorer, and select **Data** → **Edit**.

   You will see the contents of the data in the editor, as shown in Figure 6-13.

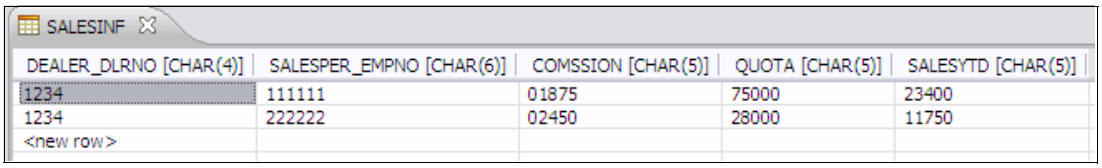


Figure 6-13 Data Studio - Data Edit

2. Edit the values, and then save the table. When you save the table, the updates, deletes, and inserts occur on the IMS database.

Restriction: In this example, the first two columns are virtual foreign key fields that cannot be updated because they are the keys of the parent segments.

Extracting data using the Data Source Explorer

You can easily extract data in a specific format, for example in a comma separated value (CSV) or an XML file, using the Data Source Explorer:

1. Right-click a table → **Select Data** → **Extract**, as shown in Figure 6-14.

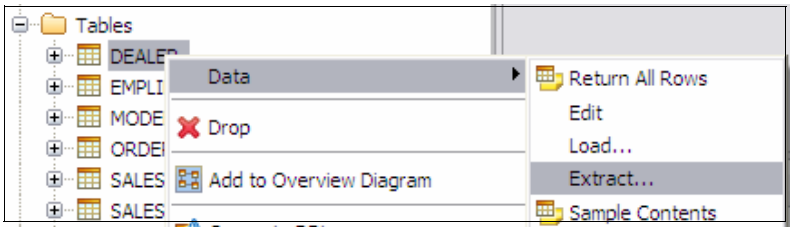


Figure 6-14 Data Studio - Extract Data

2. We specify a name, Dealer.csv, and select comma as the separator. Click **Finish**. The result is a file that contains the values of the table, as shown in Example 6-1 on page 139.

Example 6-1 Contents of Dealer.csv

```
"1235","Cupertino European Autos      ","Cupertino ","12345-6789","6667777"
"1236","IBM Bristol Cars              ","Bristol   ","BS1 6DG   ","3338888"
"8892","Thilo                         ","Stuttgart ","8888888888","888-888"
"1234","SAN JOSE FORD                 ","SAN JOSE  ","95777-3333","7774444"
"8888","Test1234                      ","Stuttgart ","8888888888","888-888"
"9999","99                            ","          ","          ","       "
```

Creating an overview diagram

You can create overview diagrams of your data with the Data Source Explorer:

Select multiple tables (hold the <CTRL> key as you click the tables), and right-click, and select **Add to Overview Diagram**, as shown in Figure 6-15.

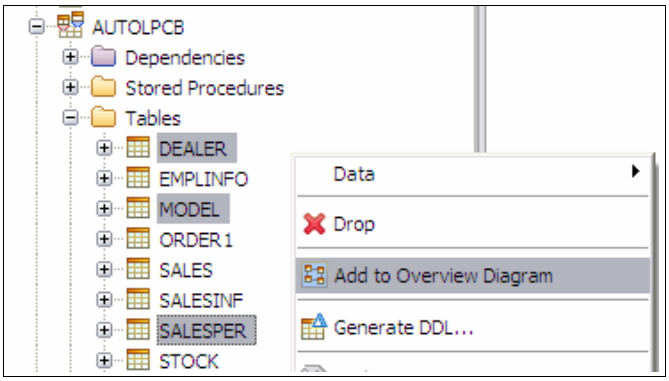


Figure 6-15 Data Studio - Add to Overview Diagram

The generated Overview Diagram is shown in Figure 6-16.

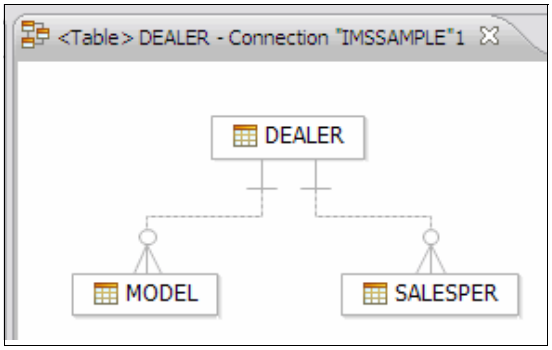


Figure 6-16 Data Studio - Overview Diagram

Working with Data Development Projects

To write your own SQL queries, you first create a new Data Development Project:

1. Click **File** → **New** → **Data Development Project**.
2. Specify the Project Name, and click **Next**.
3. Select your created IMS Data Source, and click **Finish**.

   Now you will see a new project in the Data Project Explorer.
4. Expand the Project, and right click the **SQL Scripts** folder. Select **New** → **SQL or XQuery Script**, as shown in Figure 6-17 on page 140.

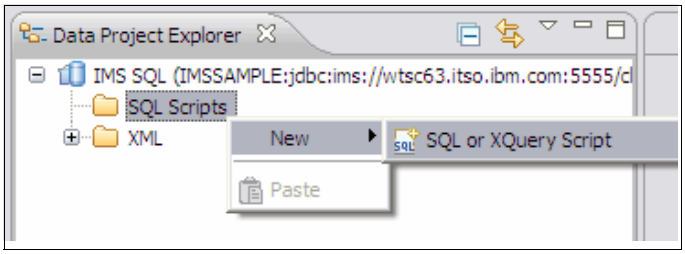


Figure 6-17 Data Studio - New SQL Script

5. You are asked what type of SQL Script you want to write. Select a single SELECT statement, as shown in Figure 6-18. Click **Finish**.

> **Tip:** The SQL and XQuery Option give you the ability to write several SQL statements separated by a semicolon, as in a batch job.

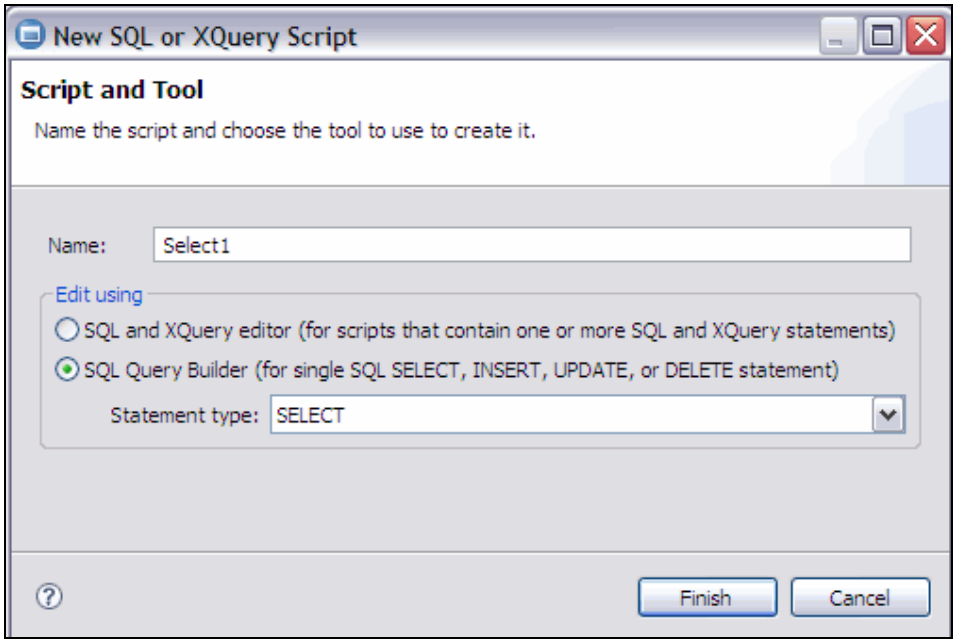


Figure 6-18 Data Studio - New SQL Script Step 2

6. To add tables, right-click the middle pane, and select tables that are in the same schema and hierarchical path for your SELECT statement. You can also specify the WHERE clause with the context help of the tool. Figure 6-19 on page 141 shows an example of a more complex SELECT statement.

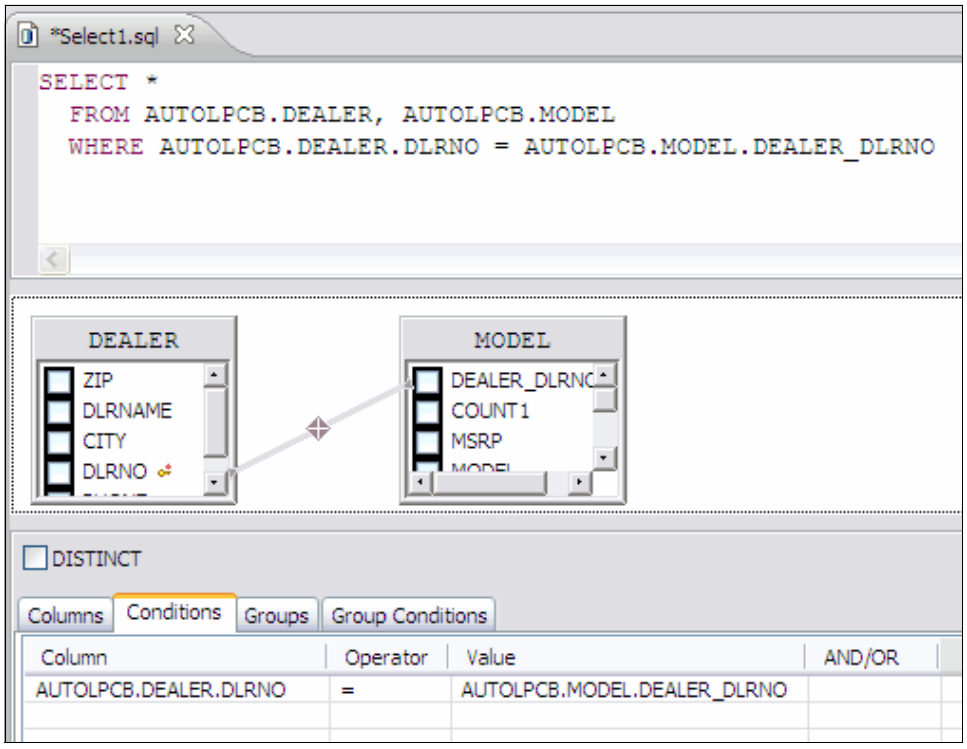


Figure 6-19 Data Studio - SQL Script Step 3

7. Save the SQL, right-click it, type `Run SQL`, and you will get the results.

There are a lot more features that you can use with the Data Perspective. There are, of course, functions that currently do not work with the IMS Universal Drivers, such as a CREATE DB statement. If you want to learn more about IBM Data Studio and its features, there are many samples in the Help feature of the product.

6.2 Accessing IMS Data in Cognos

Business needs drive the need for information. Sometimes not all information can fit into a data warehouse; moreover, some online data is often indispensable. For these reasons, you might want to retain the option to merge separate data sources in a Business Intelligence (BI) deployment.

IBM Cognos® is a Business Intelligence and Data Warehouse solution from IBM. It gives you the option to collect data from various data sources in the Cognos solution. Currently IMS Data access is not supported as a direct data source. Up to IMS 10, to be able to access IMS data, you had to either replicate the IMS data to another Data Source, such as DB2, to access it or you needed a Federation Server to access IMS databases directly. Now, with IMS 11 and the new IMS Universal JDBC Drivers, there is an easy way of getting data directly from IMS. In this section, we explain which components are used and how you can get the direct access working.

IBM Cognos 8 BI can deliver data from various sources by accessing data sources through IBM Cognos direct access and federating data sources through InfoSphere Federation Server or by using IBM Cognos Virtual View Manager. For accessing IMS data, we used the Cognos Virtual View Manager.

6.2.1 IBM Cognos 8 Virtual View Manager

IBM Cognos Virtual View Manager enables data source federation inside the BI environment. As an Enterprise Information Integrator, it provides benefits in terms of connectivity (by way of JDBC) and improves performance in multiproduct joins.

Virtual View Manager is the primary tool that is used to access multiple data sources and define, publish, and manage resources. Using Virtual View Manager you can:

- Create, edit, and manage data sources, transformations, views, SQL scripts, parameterized queries, packaged queries, definition sets, triggers, Virtual View Manager databases, and Web Services.
- Publish data sources, transformations, views, SQL scripts, parameterized queries, packaged queries, definition sets, triggers, Virtual View Manager databases, and Web Services.
- Archive Virtual View Manager resources and deploy them back to a desired location with the export/import options. This function is based on Composite Information Server technology, and it delivers standard JDBC drivers to deliver high-performance connectivity.

More information about Virtual View Manager software environments is at the following web address:

http://www.ibm.com/support/docview.wss?rs=3442&uid=swg27014427

Virtual View Manager creates views of the database that are optimized for IBM Cognos 8, and Framework Manager is then used to model the database view and create a single business view. IBM Cognos 8 components, including Framework Manager, use an Open DataBase Connectivity (ODBC) interface to access a Virtual View Manager data service. The Virtual View Manager Server accesses the data sources through Java Database Connectivity (JDBC), a Java API, ODBC, the OS File System, or SOAP, as shown in Figure 6-20.

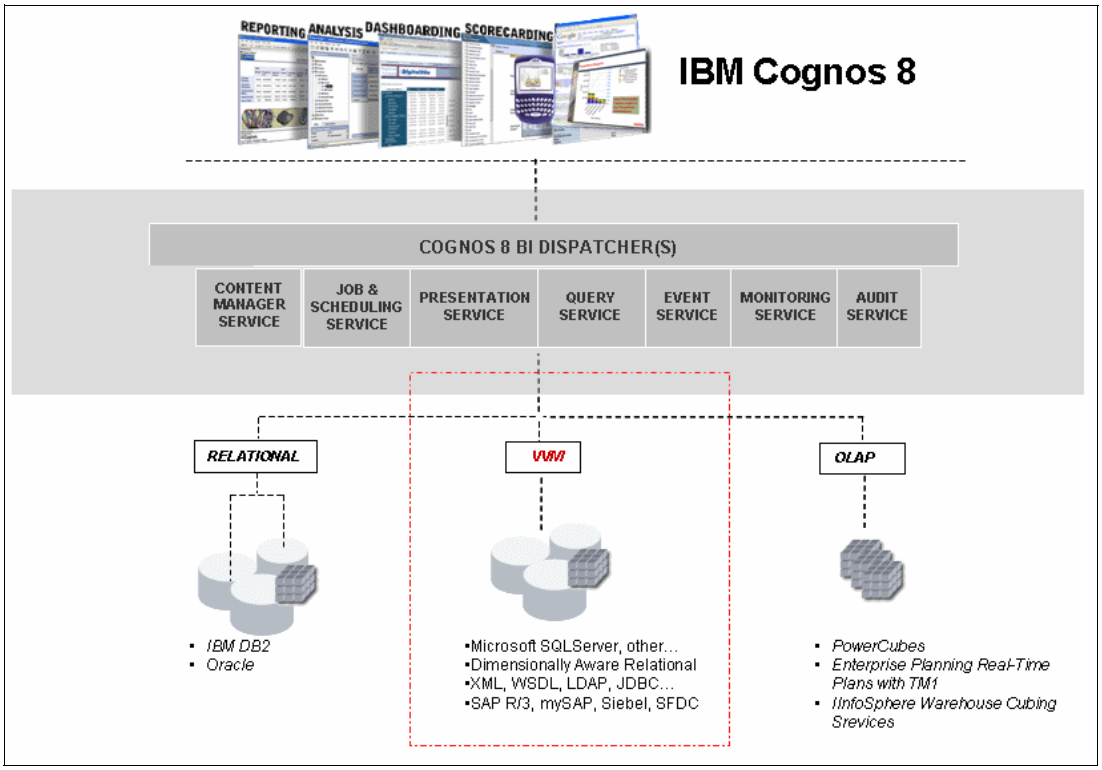


Figure 6-20 Virtual View Manager in the IBM Cognos 8 architecture

Virtual View Manager is composed of two main components: Virtual View Manager Server, which is the main engine that runs processes, and Virtual View Manager Studio, which is a client console that is used to connect, model, and expose data sources.

Basic concepts of Virtual View Manager

The workflow for using Virtual View Manager is separated into three processes:

- Setting up the environment by installing and configuring the appropriate software and drivers. The installation also installs IBM Informix® and creates a repository to contain your Virtual View Manager content.
- Creating a data source using Virtual View Manager Studio, which includes accessing and simplifying the metadata using Virtual View Manager Server.
- Accessing Virtual View Manager views using IBM Cognos 8 and preparing metadata for reporting in IBM Cognos 8.

Virtual View Manager Server can be installed in the same environment as Cognos 8, but an installation on a separate computer gives better performance and availability. The installation creates a Virtual View Manager repository and starts both the Virtual View Manager process and the Virtual View Manager Server.

Because IBM Cognos 8 uses the Virtual View Manager ODBC driver to access Virtual View Manager data sources, the ODBC driver and driver manager must be installed on each instance of the IBM Cognos 8 report server, as shown in Figure 6-21, and the Framework Manager. The driver manager routes all IBM Cognos 8 requests to the appropriate ODBC driver to access the data sources. When you add an ODBC Data Source Name (DSN) using the ODBC Data Source Administrator, you identify an ODBC driver for the driver manager. The driver manager then knows that the data source that is associated with this DSN is accessed through a particular ODBC driver.

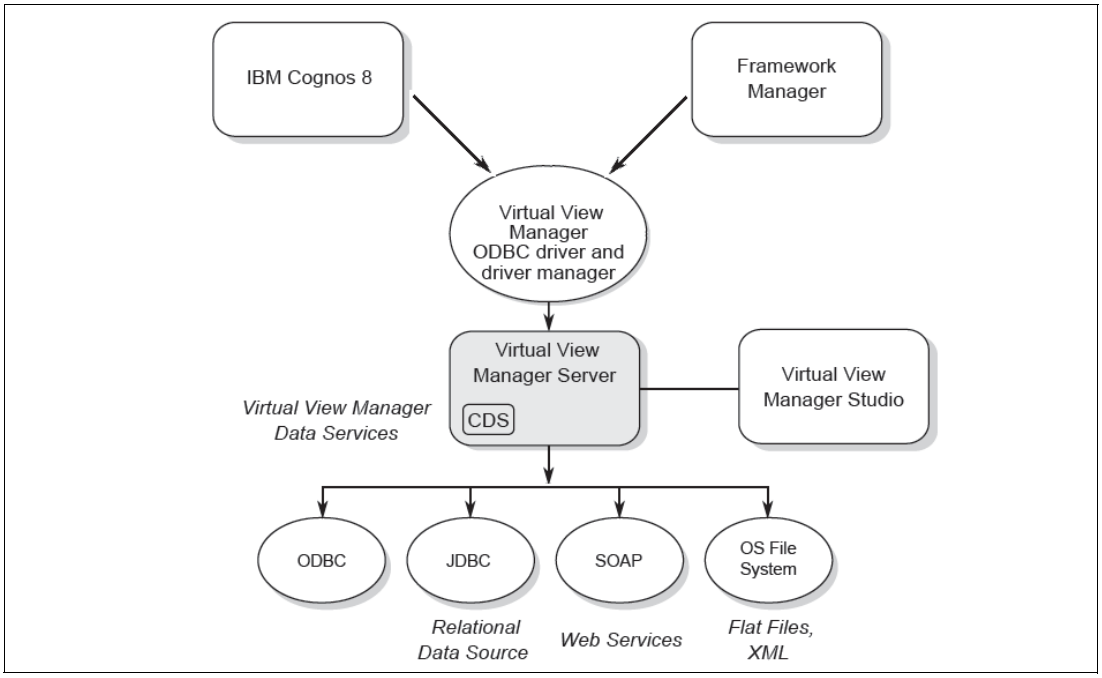


Figure 6-21 IBM Cognos Virtual View Manager architecture

Virtual View Manager Studio

Virtual View Manager Studio serves as a data view design and development area. It provides three important functions: resource management, modeling, and publishing. Figure 6-22 shows the Virtual View Manager Studio interface.

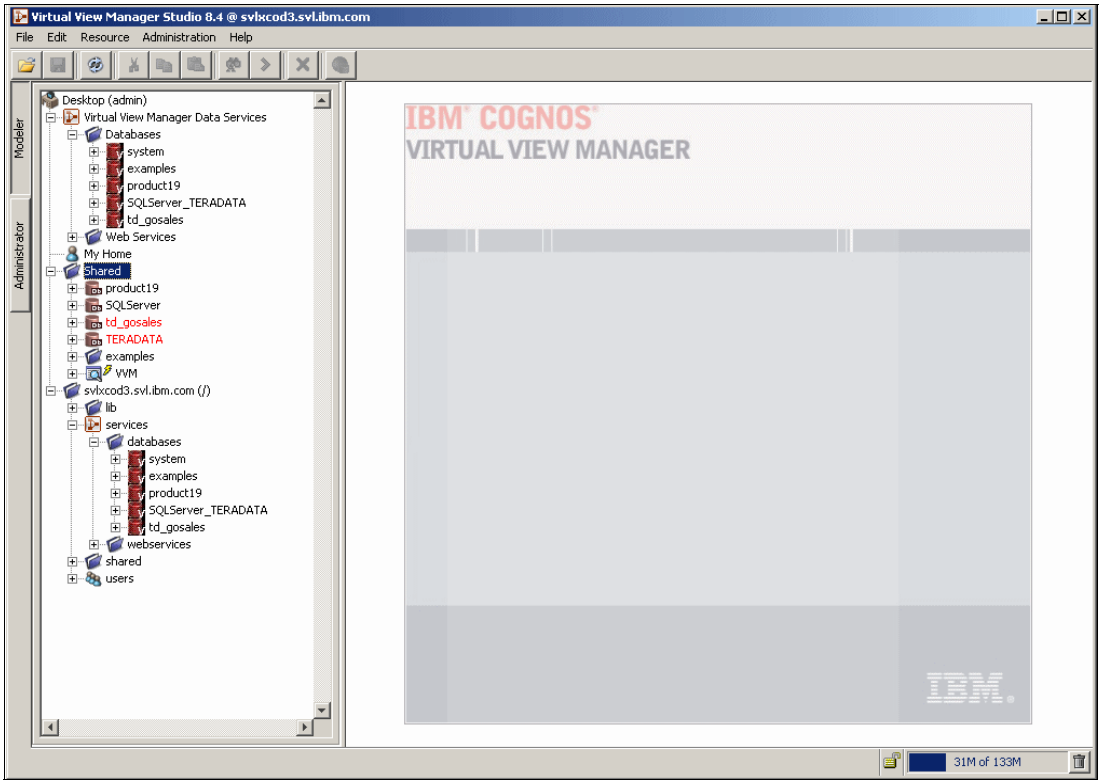


Figure 6-22 Virtual View Manager Studio interface

Modeling a Virtual View Manager data source

Use ODBC to access the Virtual View Manager data source from IBM Cognos 8 using the command-line utility named driverConfig to add an ODBC DSN. The data source associates a particular ODBC driver with the data that you want to access through that driver. To be able to create an ODBC DSN, the user that publishes the views must have the appropriate permissions for the Virtual View Manager configuration files and libraries. The ODBC DSN must be configured with the same parameters on the client in which Cognos 8 Framework Manager is used to model and publish BI contents.

In IBM Cognos 8 BI, a data source is a named set of connections to a physical database or other data source. Cognos 8 BI connects to Virtual View Manager data sources using an ODBC data source connection. Data source authentication is needed when adding a data source.

6.2.2 Configuring Virtual View Manager for IMS Data access

In this example, we show how to configure IBM Virtual View Manager (VVM) 8.4.1 for Windows with the Virtual View Manager Studio on your workstation.

Note: Because VVM currently uses JRE Version 1.5, the Java Metadata files also must be JRE 1.5 compatible; otherwise, you get a Java unsupportedClassVersion Exception.

To configure Virtual View Manager for IMS Data access:

1. Copy the IMS Universal JDBC driver and the IMS DB Metadata Jar files to the correct path so that VVM can make use of them:
   a. Create a directory called ims_universal_jdbc_driver in the conf\adapters\custom directory of your VVM installation directory. In our case, this is:

      ```
      C:\Program Files\cognos\vvm\conf\adapters\custom\ims_universal_jdbc_driver
      ```

   b. Copy the IMS Universal JDBC driver (imsudb.jar) and the IMS DB Metadata Jar file (in this case: AUTPSB11.jar) to this directory.

   > **Note:** Alternatively, you can put the driver and metadata files in the JRE lib/ext folder, which in this case is C:\Program Files\cognos\vvm\jre\lib\ext. However, avoid this when possible because it is not a good usage paradigm.

2. Start the VVM Server by selecting the **Windows start menu** → **Programs** → **IBM Cognos Virtual View Manager** → **Server** → **Server start**.
3. When the Server is started, start the VVM Console by selecting the **Windows start menu** → **Programs** → **IBM Cognos Virtual View Manager** → **Studio** → **Studio**.
4. You can now login with the user ID and password (default is admin/admin), as shown in Figure 6-23.

Figure 6-23 VVM: Login panel

5. In the VVM Studio, you can right-click the **Shared** Directory, and select **New Data Source**, as shown in Figure 6-24 on page 146.

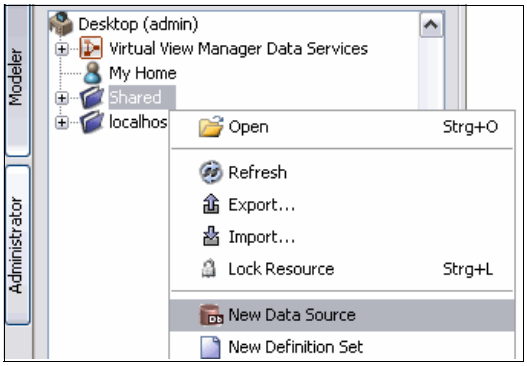


Figure 6-24 VVM: New Data Source - Step 1

6. In Figure 6-25, click **New Adapter**.

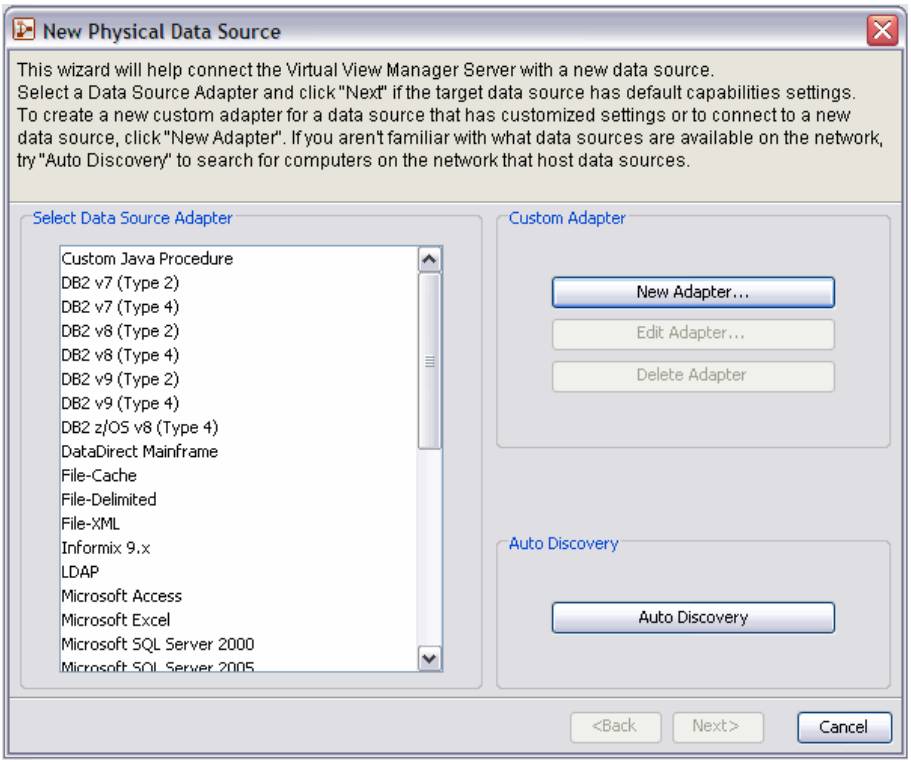


Figure 6-25 VVM: New Data Source - Step 2

7. Insert the values of Table 6-1, as shown in Figure 6-26 on page 147.

Table 6-1 New Adapter Values

Attributes	Value
Name	IMS Universal JDBC driver
Parent Adapter	Generic JDBC
Adapter Class Name	com.ibm.ims.jdbc.IMSDriver
Connection URL Pattern	jdbc:ims://<HOST>:<PORT>/<DATABASE_NAME>

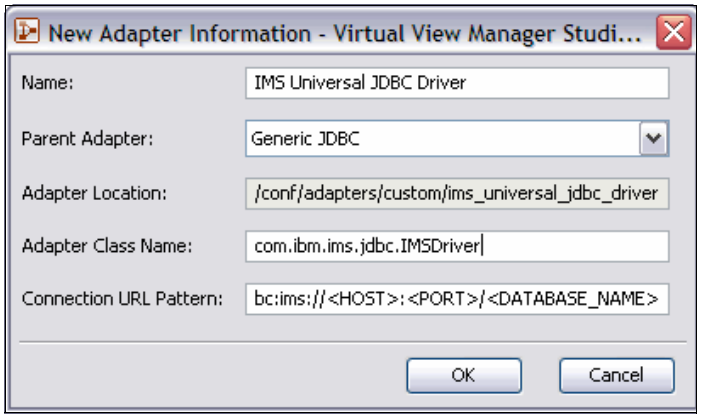


Figure 6-26 New Adapter values

8. Click **OK**, select the newly created Driver from the list, and click **Next**. The following window, Figure 6-27, is displayed. Specify your IMS Connect host name or IP Address, Port number, the fully qualified name of the metadata that is used to access your database, User ID, and Password, if security in IMS is enabled.

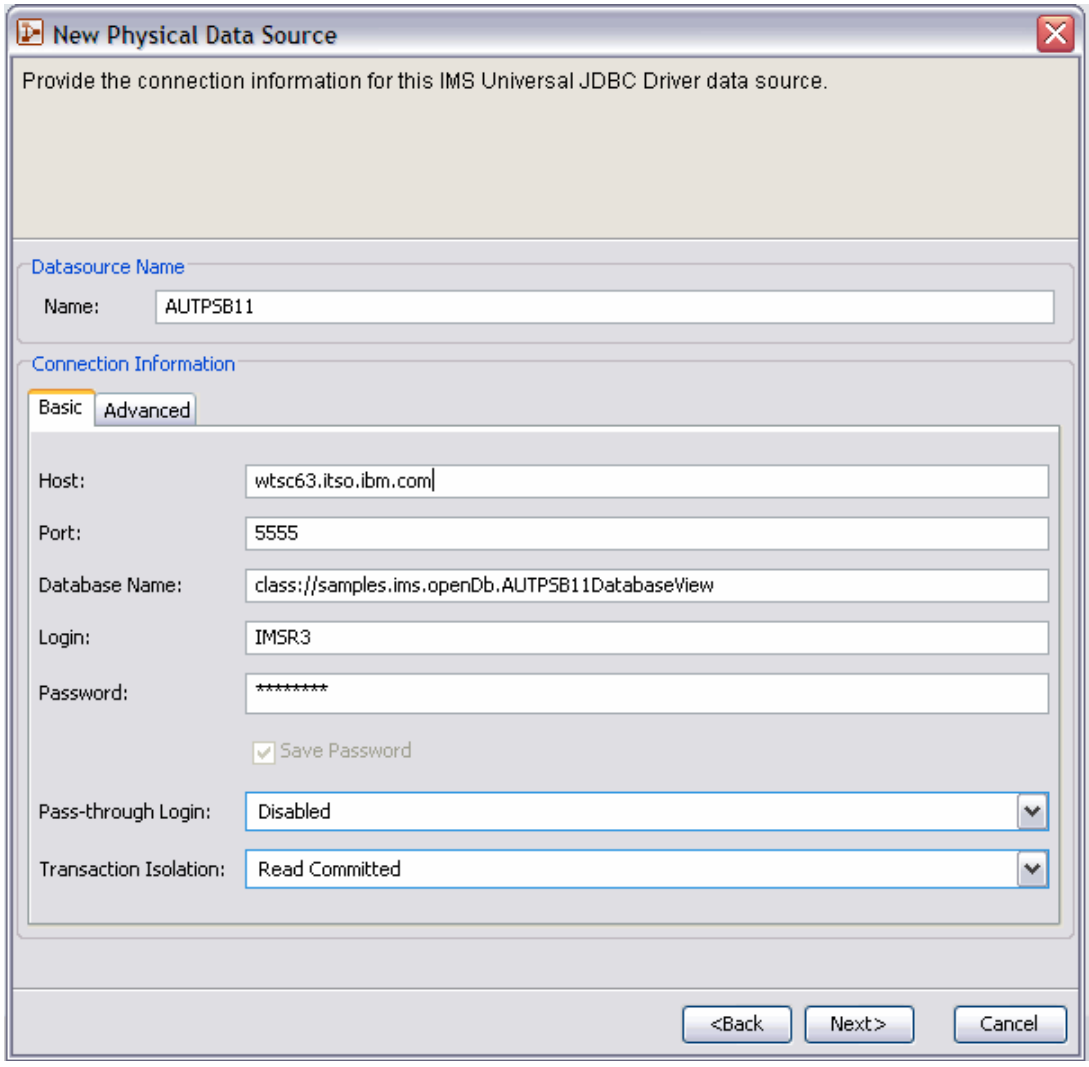


Figure 6-27 VVM: New Data Source Step 3

9. Switch to the Advanced tab, as shown in Figure 6-28.

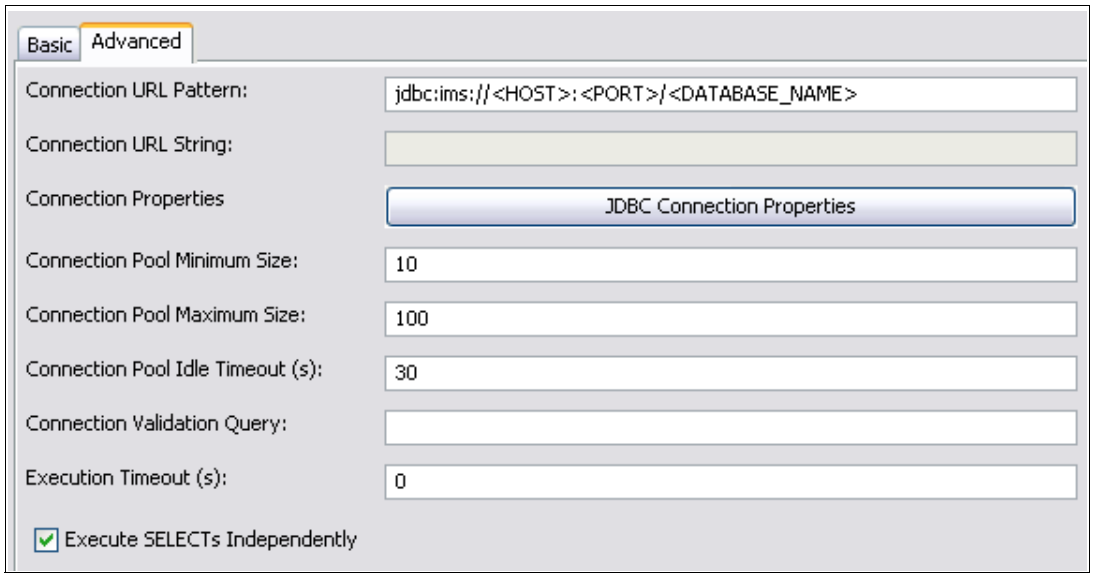


Figure 6-28 VVM - New Data Source Step 4

10. Select **Execute SELECTs Independently**, and click **JDBC Connection Properties**. Specify the two connection properties of Table 6-2, as shown in Figure 6-29.

Table 6-2 JDBC Connection Properties

Property	Value
dpsbOnCommit	true
fetchSize	100000

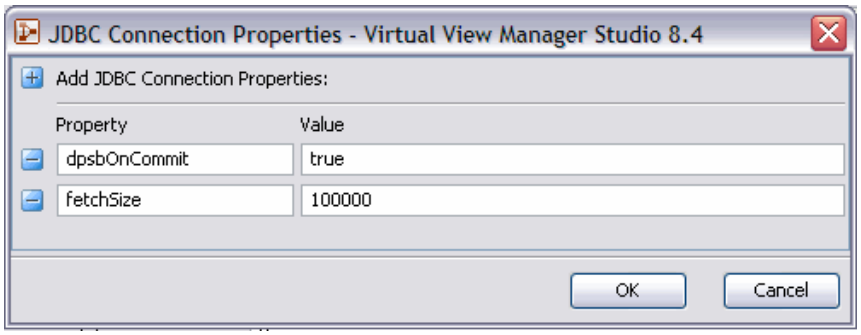


Figure 6-29 JDBC Connection Properties

The dpsbOnCommit value helps the IMS Universal JDBC driver to keep the connection alive in environments where pooled connections are used without notifying the Driver.

The fetchSize value specifies the number of rows that are collected during one network call. Increasing this parameter influences performance, if you do queries with many results.

11. Click **OK**, and then click **Next**. In the following panels, you can specify the tables that you want to select.

12. You can query results to test the access to the database by right-clicking a table and selecting **Show Contents**, as shown in Figure 6-30.

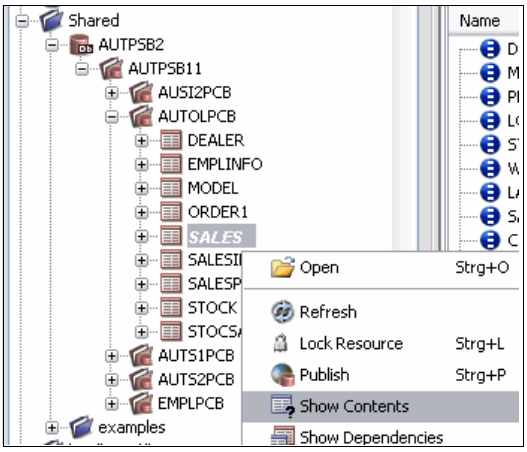


Figure 6-30 VVM - Show Contents of a table

After the connection is successful, you can build an ODBC Connection for Cognos by following the steps on the web site:

http://publib.boulder.ibm.com/infocenter/c8bi/v8r4m0/index.jsp?topic=/com.ibm.swg.im.cognos.vvm_installation_guide.8.4.0.doc/vvm_installation_guide_id1191AddSystemDataSource.html.

6.3 Accessing IMS Data using the IBM Mashup Center

IMS Web 2.0 support of IBM Mashup Center V2.0 enables you to create RSS, Atom, or XML feeds from IMS databases using the IMS Universal DB resource adapter in IMS Version 11.

With IBM Mashup Center V2.0, you can extend IMS data into Web 2.0 solutions through standard SQL queries. IBM Mashup Center has a graphical user interface for creating, customizing, and transforming feeds.

IBM Mashup Center also includes a mashup builder to enable web-savvy business users to mash information from various sources into a single view of disparate sets of information and create a flexible and dynamic application.

If you do not already have the IBM Mashup Center installed, a 60-day trial Windows-based version, which includes all product features, can be downloaded from:

http://www.ibm.com/software/info/mashup-center/

Included with the IBM Mashup Center is IBM WebSphere Application Server, Version 7.0.0.5.

Also required is the IMS Enterprise Suite DLIModel utility plug-in, which is needed to generate a metadata file that describes the database view. We are using the Java metadata class file AUTPSB11.jar generated in Chapter 4, “Generating IMS metadata class with the IMS Enterprise Suite DLIModel Utility” on page 77.

First you must decide on the database connection approach. If you plan on using the JNDI (managed) connection type, where WebSphere Application Server manages the connection to the IMS database:

1. Copy the IMS Universal JCA/JDBC resource adapter imsudbJLocal.rar to a location that is accessible to WebSphere Application Server. To install the IMS Universal DB resource adapter:
   a. From the IBM Mashup Centre v2.0 start menu, select and open the WebSphere Application Server administrative console.
   b. In the WebSphere Application Server administrative console, click **Resources** → **Resource Adapters**, as shown in Figure 6-31.

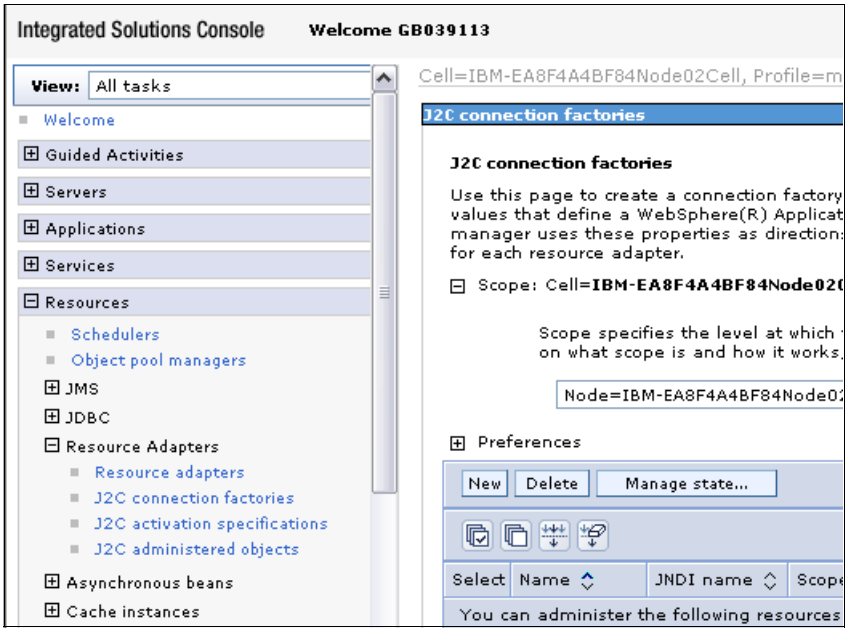


Figure 6-31 Install RAR

   c. Click **Install RAR**. The Install RAR File page is displayed.
   d. Type the path to the imsudbJLocal.rar file or click **Browse** to navigate to and select this RAR file. The path can be to a local location or to a location on the server.
   e. Click **Next**. The configuration page opens.
   f. Click **OK**. The IMS Universal DB resource adapter - JDBC Local Transaction resource adapter is listed.
   g. In the messages box, click **Save**.
2. Configure a connection factory and specify a JNDI name in the WebSphere Application Server:
   a. In the WebSphere Application Server administrative console, click **Resources** → **Resource Adapters** → **J2C connection factories**. See Figure 6-32 on page 151.

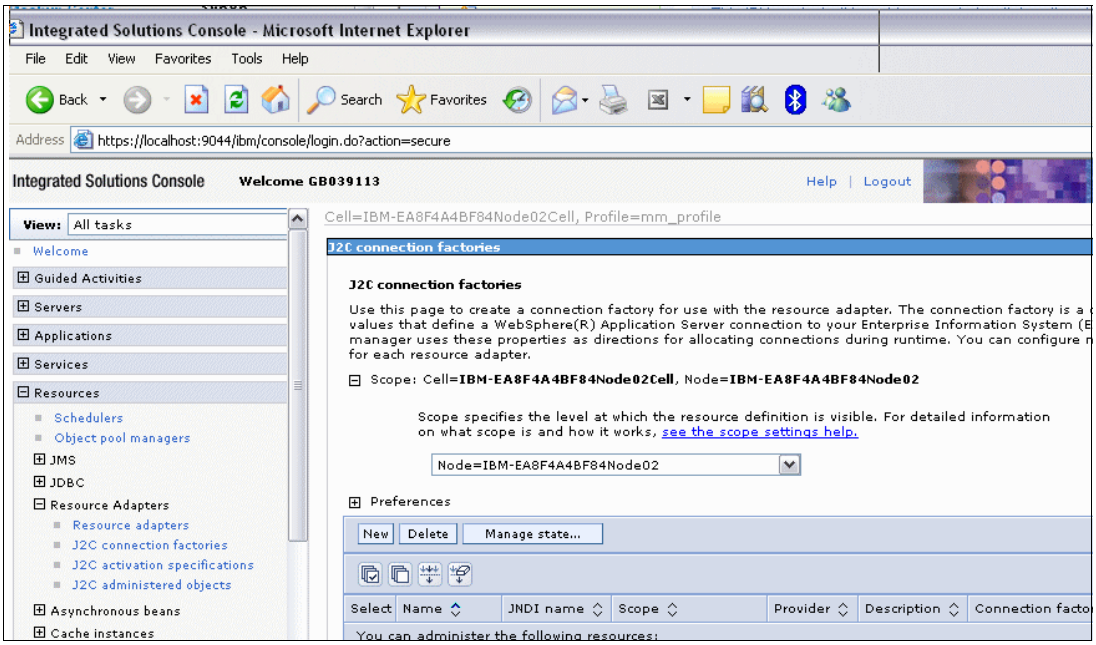


Figure 6-32 Selection of J2C connection factories

b. Click **New**.

c. From the list provided, select the name of the IMS Universal DB resource adapter that you configured.

d. Type a descriptive name for this connection factory.

e. Type the JNDI name for this connection factory.

f. Under Additional Properties, click **Custom properties**.

g. Type the information for the following properties:

- DatastoreName
- DatastoreServer (the IMS host system)
- MetadataURL
- Password
- PortNumber
- User

h. Click **Apply**.

i. In the messages box, click **Save** to save this configuration information.

3. Copy the compiled metadata package that you generated into the <install-dir>\AppServer\profiles\profileName\installedConnectors\imsudbJLocal.rar\ directory.

4. Stop and restart the WebSphere Application Server.

If you plan to use the Driver Manager (non-managed) connection type:

1. Provide the IMS data store name, IMS Connect port number, username, and password for connecting to the database and the fully qualified name of the Java metadata file (metadata URL) that was generated by the IMS Enterprise Suite DLIModel utility plug-in.

2. Copy the metadata package that you generated into <install-dir>\AppServer\lib directory.

3. Stop and restart the WebSphere Application Server.

To create an IMS feed:

1. Select and start the InfoSphere MashupHub from the IBM Mashup Centre V2.0 menu.
2. On the Home:Catalog tab, click **Create**, and select **New Feed**, as shown in Figure 6-33.

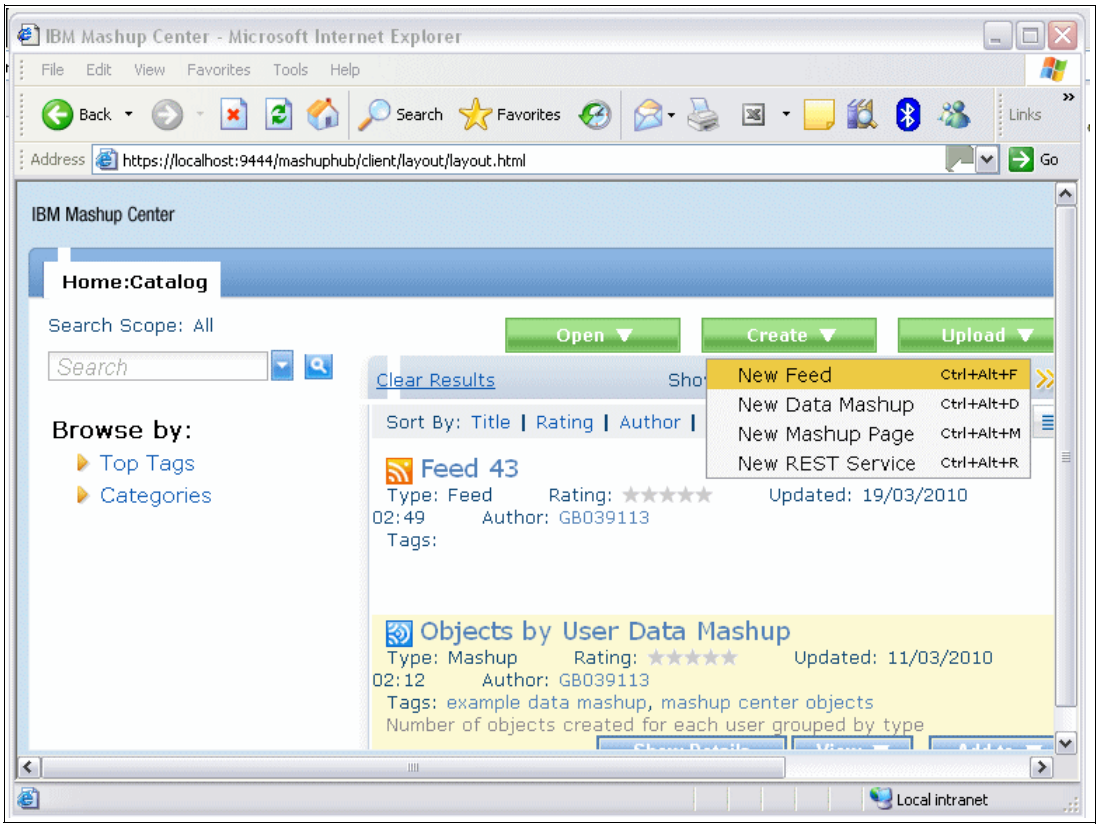


Figure 6-33 Creating a new Feed

3. Select **Enterprise database (JDBC)**, and click **Next**, as shown in Figure 6-34.

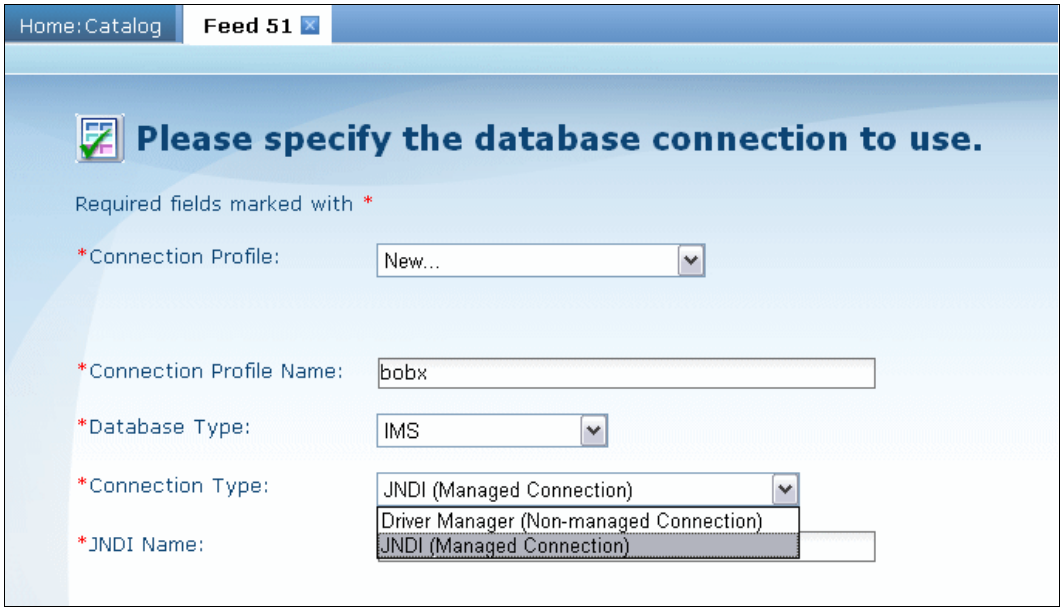


Figure 6-34 Database connection panel

4. In the Select field, select **New** to create a new database connection profile.
5. In the Profile Name field, type the new profile name.
6. In the Database Type field, select **IMS**.
7. In the Connection Type field, select either the Driver Manager (Non-managed Connection) or JNDI (Managed Connection) and fill in the rest of the required information. If you are using the JNDI connection, the required JNDI Name is the name of the JNDI name that you just created.
8. Click **Next**. The SQL Query Builder opens, as shown in Figure 6-35.

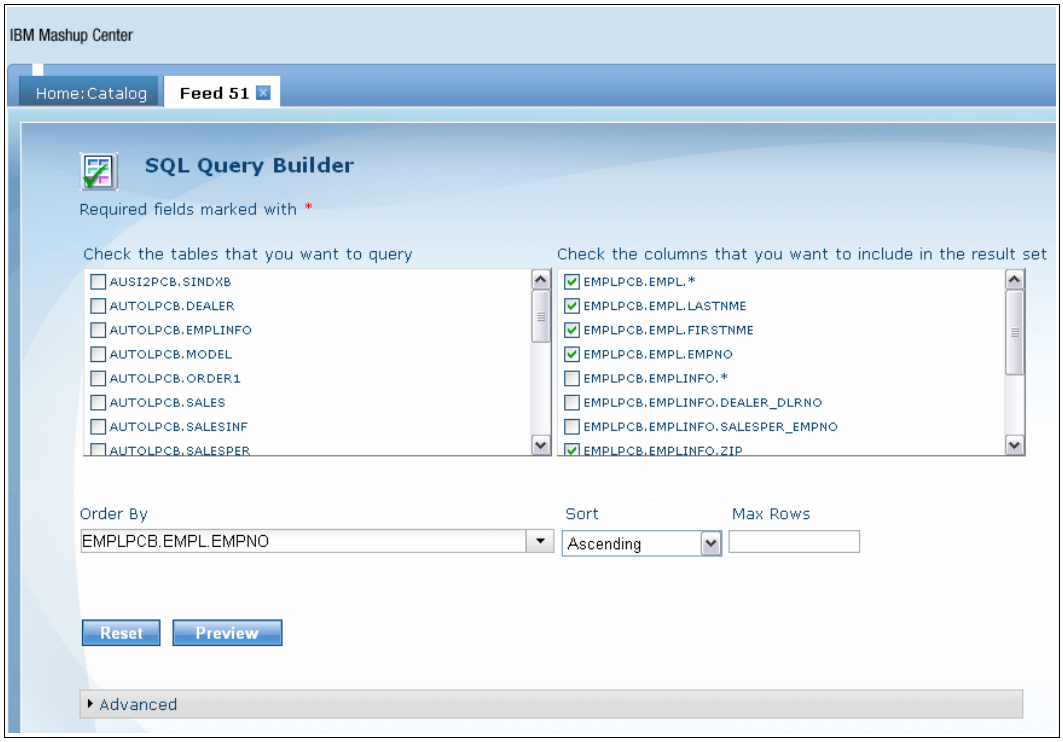


Figure 6-35 The SQL query builder panel

9. In the SQL Query Builder, create the SQL query by selecting the tables and columns to construct a basic query. If customization or a more advanced SQL query is required:
   a. Click **Generate SQL** to get the query that is associated with your selection of tables and columns.
   b. Click to expand the Advanced field to type or customize your SQL statement.
   c. Click Preview to see the results of your query in a new window.
   d. See the online help for assistance and guidance information about how to formulate your SQL statement.
10. When you are finished with the SQL query, click **Next**, as shown in Figure 6-36 on page 154.

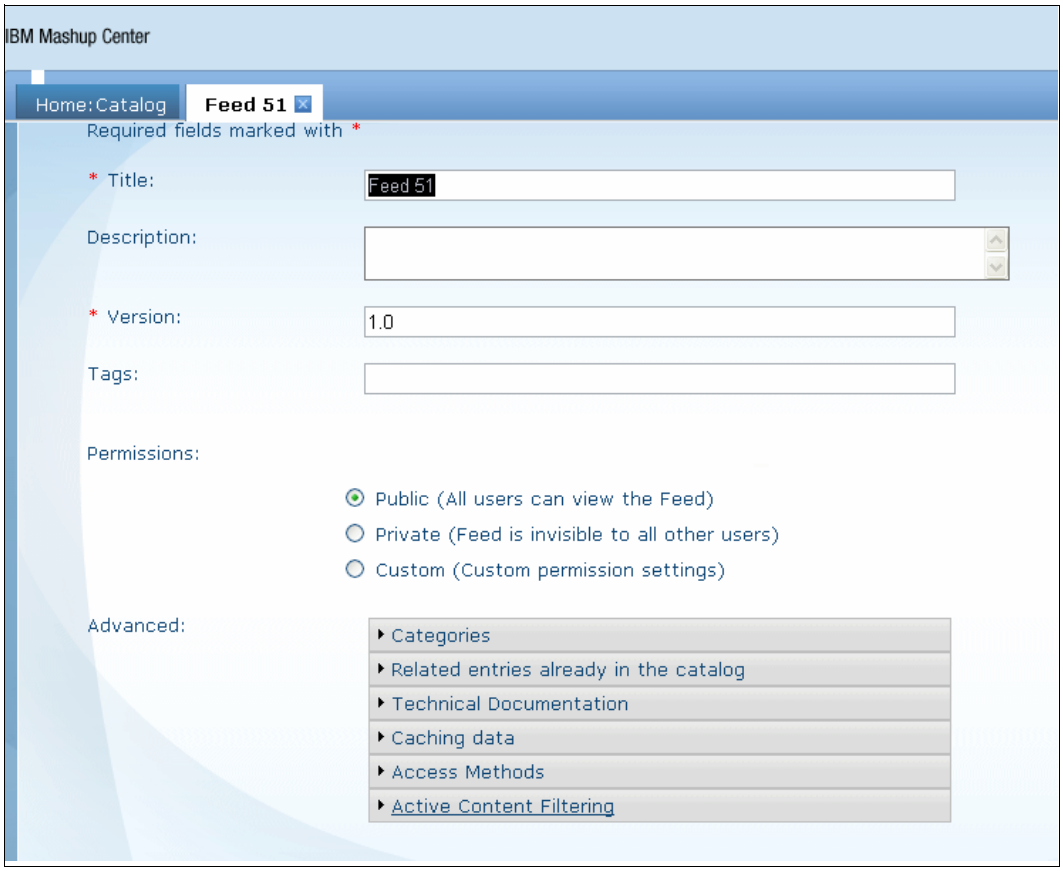


Figure 6-36 The final feed panel

11. Specify the required information to identify this feed. All fields with an asterisk (*) are required.
12. When you are finished specifying the information for this feed, click **Finish**. You are notified that the information is saved.

Now you can use the feed in the Mashup Hub, for example, to combine an IMS feed that displays the address of a car dealer with a map of the location of the dealer that you selected. With this approach, it is very easy to generate valuable web site data that is already available.

7

Scenario 2: Developing JDBC applications

In this chapter, we focus on application development with the IMS Universal JDBC Drivers in managed and non-managed environments. We explain, in detail, the steps that are required when using an eclipse-based product. Rational Developer for System z with Java is used for:

- Developing a stand-alone Java application using the IMS Universal JDBC driver
- Developing a managed Java application using the IMS Universal Database Resource Adapter (XA) and DB2 Data Server Drivers (XA)
- Developing an IMS Java Transaction using the IMS Universal JDBC driver

7.1 Developing a stand-alone Java application using the IMS Universal JDBC driver

This scenario shows you how to develop a Java application step-by-step for a non-managed environment accessing the IMS Car Dealer IVP Database using the IMS Universal JDBC driver.

7.1.1 Prerequisites

The prerequisite products for this scenario are:

- IDE with Java Development Kit 1.5

  In this scenario, we use IBM Rational Application Developer (RAD) as the integrated development environment (IDE) for Java.

  If you want to try Rational Application Developer, a Download Trial Version is available from the IBM web site at

  `http://www.ibm.com/developerworks/downloads/r/rad/`

- IMS Universal JDBC driver

  You need the IMS Universal driver imsudb.jar, which is the stand-alone JDBC and DL/I Driver for IMS.

- Metadata Java class

  In our example, we use the Metadata for an IMS database that is normally installed as part of the Installation Verification Procedure (IVP) of IMS. It is a Car Dealer Database and uses a PSB called AUTPSB11. The IMS Enterprise Suite DLIModel Utility generated a class file for this PSB. It is exported to the Jar File AUTPSB11.jar. The full qualification of the metadata class file is: class://*samples.ims.openDb.AUTPSB11DatabaseView*

- IMS with IMS Connect

  IMS and IMS Connect and the Open Database Manager address space must be set-up correctly for use of the IMS Open Database feature. You need the host name or IP address, the Portnumber where IMS Connect is listening for Open Database Connections, and the Data Store Name or the Alias Name (if specified) of the IMS you want to access. If security is enabled in IMS Connect, you also need a valid user ID and password.

7.1.2 Creating and configuring a new Java project

To create and configure a new Java project:

1. Start the Rational Application Developer, and select your workspace.
2. Open the Java Perspective by clicking the **Switch Perspective** icon and selecting **Java**, as shown in Figure 7-1 on page 157.

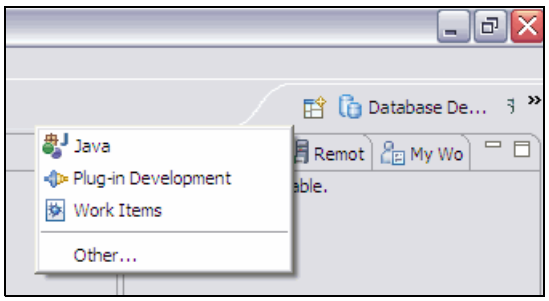


Figure 7-1 Switch to Java perspective

3. Select **File** → **New** → **Java Project**.
4. Specify the name of the Java Project, and leave the default settings as they are. Click **Finish**.
5. Add the IMS Universal JDBC driver and the Java Metadata Class file to your project build path so that your application finds the referenced Java packages. Right-click your project in the Package Explorer View, and select **Build Path** → **Configure Build Path**, as shown in Figure 7-2.

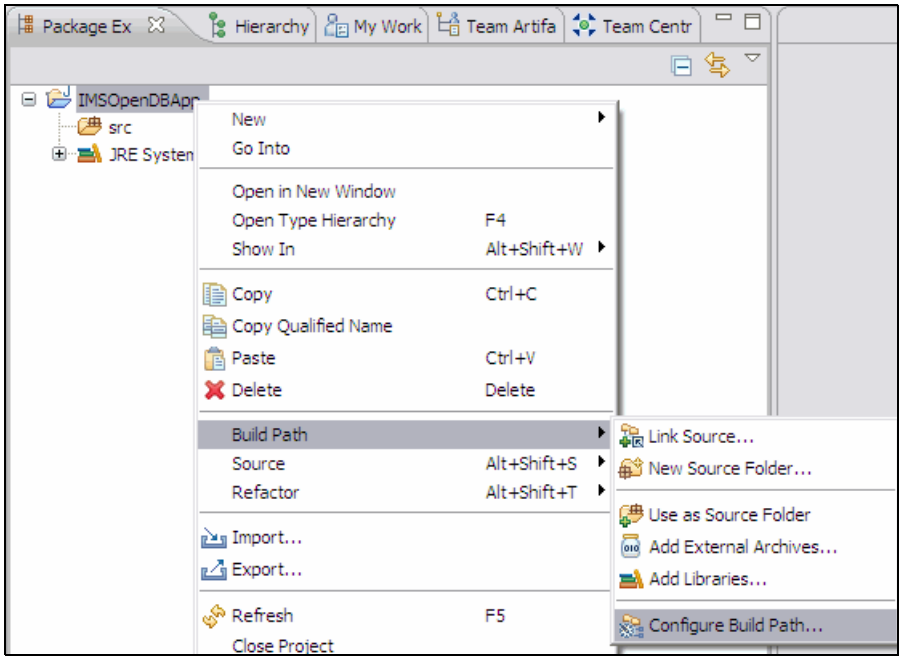


Figure 7-2 Configure Build Path

6. Click the **Libraries** tab, and click **Add external JARs**. Select the **imsudb.jar** on your hard disk.
7. Repeat the last step, and also add the Java Metadata jar file that contains the IMS Database Metadata class file called **AUTPSB11.jar** to the Build Path libraries.

Note: As an alternative you can also add the Metadata source file to the project in the appropriate package. However, it is better to separate these. If they are separated, you do not have to change the application for changes to the database. You just replace the referenced Metadata Jar file that contains the new Metadata class in the application's class path.

8. Figure 7-3 shows the result. Click **OK**.

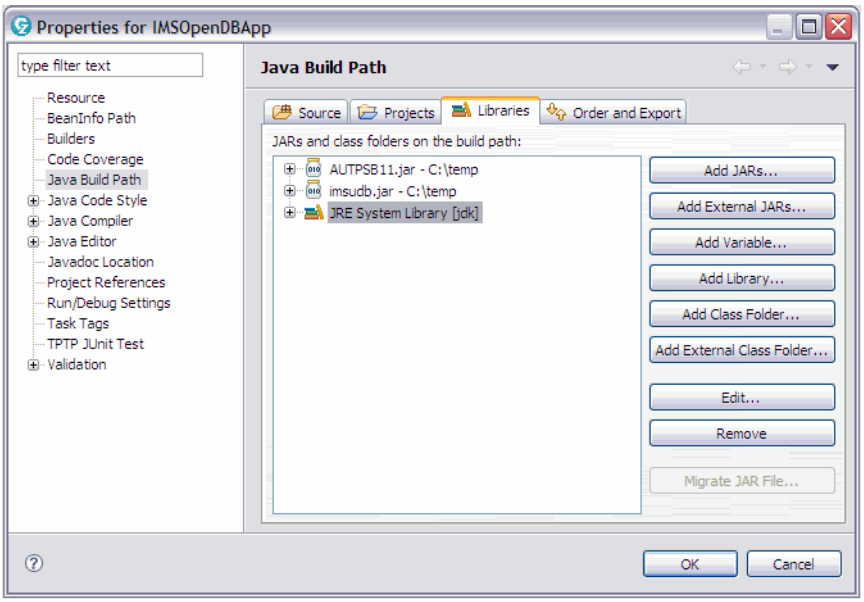


Figure 7-3 Java Build Path properties

9. Right-click your project's source folder, and select **New** → **Package**, as shown in Figure 7-4.

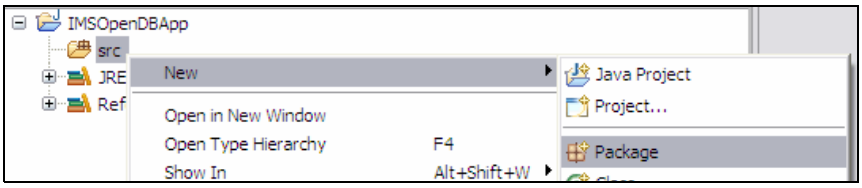


Figure 7-4 New Java Package

10. Type in a name for the package, such as `samples.ims.applications`, and click **Finish**.
11. Right-click the newly created package, and select **New** → **Class**, as shown in Figure 7-5.

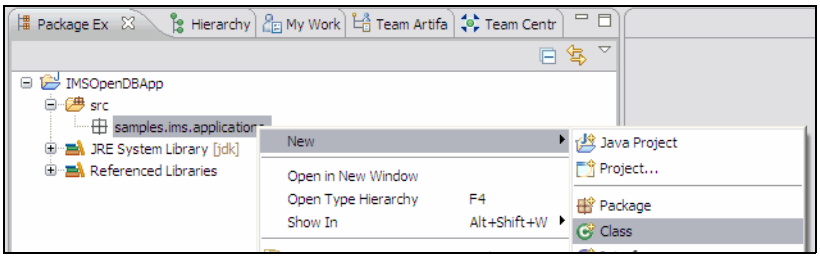


Figure 7-5 New Java Class

12. Write a name for the Java class, such as **IMSJDBCStandalone**, and select **public static void main (String[] args)** to generate a Java file with skeleton methods to define a runnable application. The other values are automatically populated, such as the package name, as shown in Figure 7-6 on page 159. Click **Finish**.

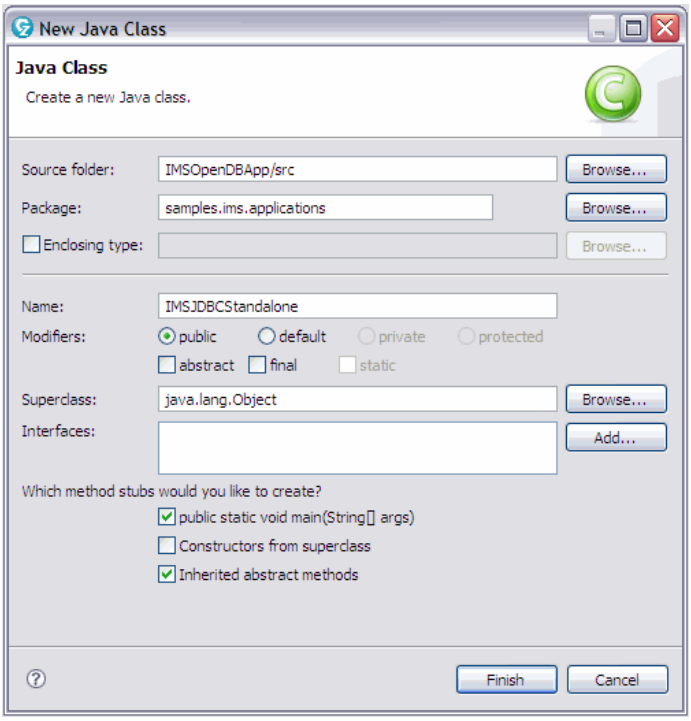


Figure 7-6 New Java Class properties

7.1.3 Writing the application

Now that you have a Java application file, you can start developing your Java application using the IMS Universal JDBC driver. The code in Example 7-1 shows a stand-alone JDBC Java application using the IMS Universal JDBC driver to do SELECT, INSERT, UPDATE, and DELETE calls.

Example 7-1 Code of IMSJDBCStandalone Application

```
package samples.ims.applications;
import java.sql.*;                                                    // 1
import com.ibm.ims.jdbc.*;

public class IMSJDBCStandalone {
   public static Connection conn;                                     // 2
   public static IMSDataSource ds;
   public static Statement st;

   public static void main(String[] args) {
      ds = new com.ibm.ims.jdbc.IMSDataSource();                      // 3
      ds.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");    // 4
      ds.setDatastoreName("IMS2");
      ds.setDatastoreServer("host.itso.ibm.com");
      ds.setPortNumber(5555);
      ds.setDriverType(IMSDataSource.DRIVER_TYPE_4);
      ds.setUser("IMSUSR");
      ds.setPassword("myPw");
      try {
         conn = ds.getConnection();                                   // 5
         conn.setAutoCommit(false);                                   // 6
         st = conn.createStatement();                                 // 7
         displayDealer();                                             // 8
```

```
                                                                          // 9
         String sql="INSERT INTO AUTOLPCB.DEALER (DLRNO,DLRNAME,ZIP,CITY,PHONE) "+
         "VALUES('8889', 'Thilo Sample Inc', '71139', 'Ehningen','555-123')";
         System.out.println("\nSQL Command: "+sql);
         int insertCount = st.executeUpdate(sql);
         System.out.println("The Number of Rows Inserted: "+insertCount);
         displayDealer();
                                                                        //10
         sql="UPDATE AUTOLPCB.DEALER SET PHONE='555-789'" +
         " WHERE DLRNO='8889' or PHONE='555-123'";
         System.out.println("\nSQL Command: "+sql);
         int updateCount = st.executeUpdate(sql);
         System.out.println("The Number of Rows Updated: "+updateCount);
         displayDealer();
                                                                       //11
         sql="DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='8889'";
         System.out.println("\nSQL Command: "+sql);
         int deleteCount = st.executeUpdate(sql);
         System.out.println("The Number of Rows Deleted: "+deleteCount);
         displayDealer();

         st.close();                                                    //12
         conn.commit();
         conn.close();
         System.out.println("\nChanges Successfully commited");
      } catch (SQLException e) {
         e.printStackTrace();
         try {                                                         //13
            conn.rollback();
            conn.close();
            System.out.println("Changes rolled back because of an error");
         } catch (SQLException e1) {                                   //14
            System.out.println("Error in Connection");
            e1.printStackTrace();
         }
      }
   }

   public static void displayDealer() throws SQLException{            //15
      String sql="SELECT * FROM AUTOLPCB.DEALER";
      System.out.println("\nSQL Command: "+sql+"\n");
      ResultSet rs = st.executeQuery(sql);                           //16
      ResultSetMetaData rsmd = rs.getMetaData();                     //17
      int numColumns = rsmd.getColumnCount();
      for (int i=1; i<=numColumns; i++) {
         System.out.print(rsmd.getTableName(i)+"."+rsmd.getColumnName(i)+" | ");
      }
      System.out.println("\n---------------------------------"+
      "-----------------------------------------");
      while (rs.next()) {                                             //18
         for (int i=1; i<=numColumns; i++) {
            System.out.print(rs.getString(i) + " | ");              //19
         }
         System.out.println();
      }
      rs.close();                                                   //20
   }
}
```

Here are the explanations for the commented numbers in the code in Example 7-1 on page 159:

1. These are the two import statements you need to use JDBC and the IMS Universal JDBC driver.
2. The application uses a function that requires access to the Connection, IMSDataSource, and the Statement variables. Therefore these variables are globally defined.
3. When the application starts, a new IMSDataSource object is created. The alternative is to obtain one from a managed container using JNDI.
4. The IMSDataSource object is populated with the required parameters using its setter methods.
5. The connection is initiated from the IMSDataSource.getConnection() function, which along with the following functions can throw SQLExceptions that must be caught and handled.
6. The application completes several interactions in one connection to be sure that all changes are committed or rolled back. Autocommit is disabled for this connection.
7. This command generates a new statement object for this specific connection.
8. Because this function is called before and after each change in the database it is separated in a function that is called.
9. The statements between comment 9 and 10 insert values in the DEALER segment of the database. For Insert, Updates, and Deletes, the Statement.executeUpdate(sql) command is used. The returned value of the method is the number of rows affected.
10. The statements between 10 and 11 update an entry inserted in the previous step that matches the DLRNO or PHONE that is specified in the WHERE clause.
11. The statements between 11 and 12 delete the inserted entry.
12. After sending the queries, the statement is closed, and the connection is explicitly committed and closed. Make sure you commit the connection before you close it; otherwise, the changes are rolled back.
13. If any Exception is thrown in the Try{...} construct, the catch(){...} construct rollsback the changes made and closes the Connection.
14. If this fails, there was an error with the connection itself, and usually your connection times out and gets rolled back on the IMS side, depending on your settings in IMS.
15. The displayDealer() function can throw an SQLException because it contains methods that throw this Exception. It is handled in the main program try{...}catch(){...} construct.
16. To issue a SELECT statement, the Statement.executeQuery(sql) function is used. It returns a ResultSet object that pulls the results from IMS depending on the fetchSize settings of the IMSDataSource object.
17. The statements between 17 and 18 show how to obtain the table and column names from the result sets.
18. The While(){} loop runs until there is no more data in the ResultSet.
19. In the DEALER segment, every field is defined as CHAR and can be pulled out as a string with the ResultSet.getString(columnName or columnIndex) method.
20. The ResultSet is closed when it is not needed anymore.

7.2 Developing a managed Java application using the IMS Universal Database Resource Adapter (XA) and DB2 Data Server Drivers (XA)

This scenario shows the major steps to set up the IMS Universal Resource Adapter in WebSphere Application Server and how to develop a managed application that accesses IMS and DB2 resources using XA in both drivers. This sample also shows that the syntax for programming against IMS and DB2 is very similar.

7.2.1 Prerequisites

To follow this scenario, you need the following prerequisite products:

- Rational Application Developer with Java Development Kit 6

  For this scenario, we use IBM Rational Application Developer as the integrated development environment (IDE) for Java because if offers good integration with WebSphere Application Server and also offers capabilities for easier development of JEE applications.

  If you want to try Rational Application Developer, a downloadable trial version is available from the IBM web site at:

  `http://www.ibm.com/developerworks/downloads/r/rad/`

- WebSphere Application Server 7.0

  WebSphere Application Server 7.0 can be used alone or as the integrated Test Environment Installation of Rational Application Developer.

  You can download a trial version of WebSphere Application Server from the IBM web site at:

  `http://www.ibm.com/developerworks/downloads/ws/was/`

- IMS Universal Database Resource Adapter

  You need the IMS Universal Database Resource Adapter called imsudbJXA.rar, which is the managed JDBC Resource Adapter with XA support.

- IMS DB Metadata Java class

  In our example, we use the Metadata for an IMS database that is normally installed as part of the Installation Verification Procedure (IVP) of IMS. It is a Car Dealer database and uses a PSB called AUTPSB11. The IMS Enterprise Suite DLIModel Utility generated a class file using this PSB as input. It is exported to a jar file called AUTPSB11.jar. The full qualification of the metadata class file is:
  class://samples.ims.openDb.AUTPSB11DatabaseView

- IBM (DB2) Data Server Driver for JDBC and SQLJ

  For accessing DB2 tables, you need the Data Server Drivers for JDBC and SQLJ for DB2. This driver is also called the JAVA Common Client (JCC) driver and was formerly known as the IBM DB2 Universal Database™ Driver. Include db2jcc.jar for JDBC 3.0 and earlier functions and db2jcc4.jar for JDBC 4.0 and later functions.

  If you are using DB2 for z/OS, you need the license file db2jcc_license_cisuz.jar.

  For more information about the DB2 drivers, see Appendix A.1, “IBM Data Server drivers and clients” on page 230.

- DB2

  In this scenario, we also access DB2 tables that are stored in DB2 for z/OS, but for the success of this tutorial, DB2 can be on any platform.

- IMS with IMS Connect, ODBM, and RRS enabled

  IMS and IMS Connect and the Open Database Manager address space must be set-up correctly for use of the IMS Open Database feature. Because we use the XA Drivers, IMS, IMS Connect, and ODBM, all of them must be configured to use RRS; otherwise, you will receive HWSK2880E RRS COMMAND FAILED … messages from IMS Connect when running the application.

  You need the host name or IP address, the port number where IMS Connect is listening for Open Database connections, and the Data Store Name or the Alias Name (if specified) of the IMS that you want to access. If security is enabled in IMS Connect, you need a valid user ID and password.

7.2.2 Installing the products

For the installation steps of the products, see the IBM Information Center for the corresponding product.

For installing WebSphere Application Server 7.0 see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.installation.base.doc/info/aes/ae/welc6topinstalling.html

For installing Rational Application Developer see:

http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/c_intro_product.html

7.2.3 Creating the projects in Rational Application Developer

In this section, we explain step-by-step how to create and set up the necessary files and projects that are needed later in Rational Application Developer:

1. Start Rational Application Developer.
2. Select **File** → **New** → **Other** → **Enterprise Application Project**, and click **Next**. Name it `IMSandDB2EAR`. In the Target Runtime field, select **WebSphere Application Server 7.0**. In the Version field, select **EAR Version 5.0**, as shown in Figure 7-7 on page 164. Click **Finish**.

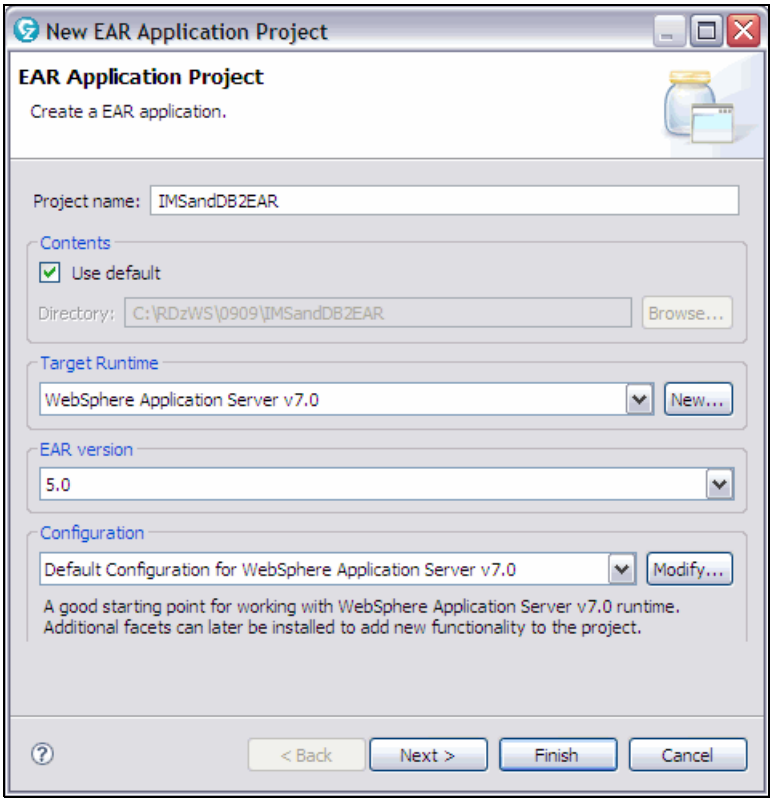


Figure 7-7 New EAR Project

3. Select **File** → **New** → **Other** → **EJB Project**. Name it `IMSandDB2EJB`. Select **Add Project to an EAR**, and select the created EAR Project. Make sure you select EJB Module version **3.0**, and click **Next**. Clear the **Generate a Client Jar** option, and select **Generate Deployment Descriptor,** as shown in Figure 7-8 on page 165. Click **Finish**.

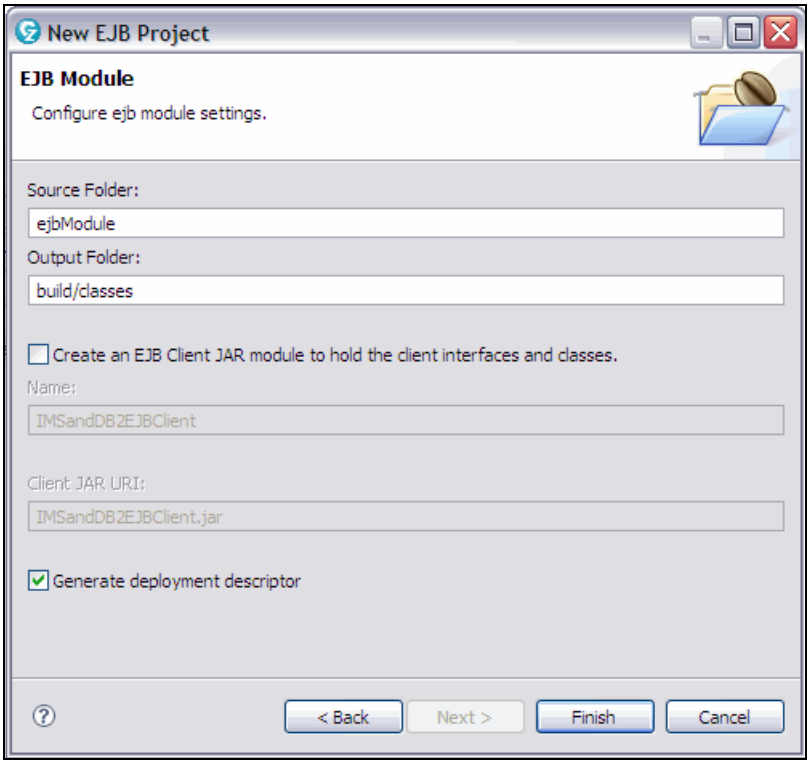


Figure 7-8 New EJB Project

4. Select **File** → **Import** → **RAR File**, and click **Next**. Select the **imsudbJXA.rar** on your hard disk, and click **Finish.**
5. Select **Window** → **Open Perspective** → **Other**, and select **Java EE**. Click **OK**.
6. Right-click the EJB Project, and select **Build Path** → **Configure Build Path.** Switch to the **Projects tab**, add the imported imsudbJXA, and click **OK**.
7. Select **File** → **New** → **Other** → **Session Bean**, and click **Next**. Select your created EJB Project, and name the bean `XASessionBean` in the Java Package samples.ims, as shown in Figure 7-9 on page 166. Click **Finish.**

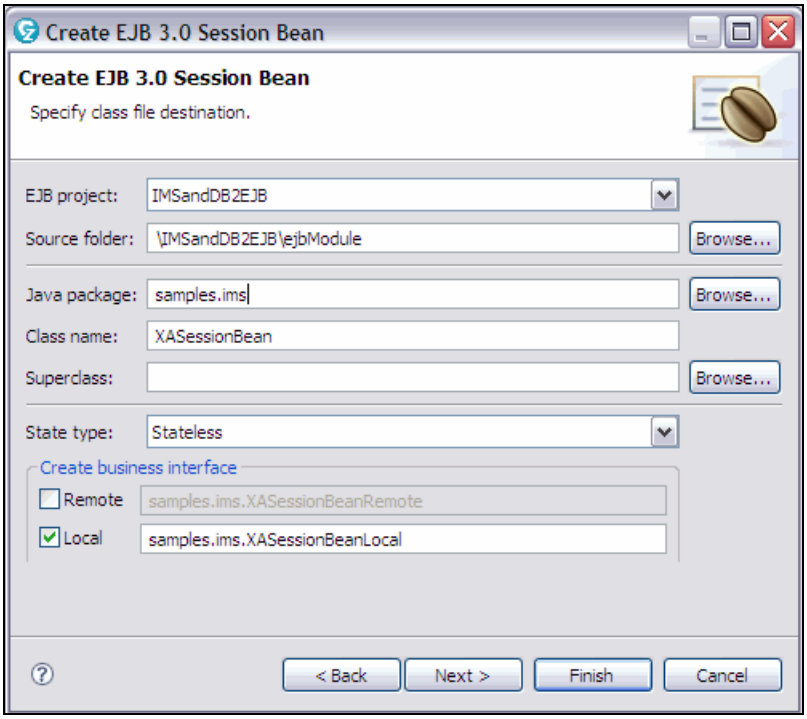


Figure 7-9 New EJB 3.0 Sessionbean

8. Select **File** → **New** → **Other** → **Dynamic Web Project**, and click **Next**. Name it `IMSandDB2Web`, select **Add Project to an EAR**, and select the created EAR Project. Select Dynamic Web Module version **2.5**, and click **Finish**.
9. Right-click the web project, and select **Build Path** → **Configure Build Path.** Switch to the **Projects tab**, and add the IMSandDB2EJB project. Click **OK**.
10. Select **File** → **New** → **Other** → **Servlet**, and click **Next**. Select your created web project, and name the servlet `XAServlet` in the Java Package samples.imsservlet. Click **Finish.**
11. Select **File** → **New** → **Other** → **Web page**, and click **Next**. Select your created web project, name the web page `XATest.jsp`, and click **Finish.**

7.2.4 Sample code for a managed environment

After setting up the required Projects and Build Paths, you can start to write the application. In this section, we provide an example of coding that allows you to issue SQL commands against IMS and DB2 using the XA Drivers for both components and the managed JDBC programming approach. The EAR Project does not need to be changed. You must modify the EJB and the EJB Interface in the EJB project and the servlet and the JSP web site in the web project.

XASessionBeanLocal Interface

Example 7-2 shows the sample code for the Java EJB files in the EJB Project.

Example 7-2 XASessionBeanLocal.java

```
package samples.ims;
import javax.ejb.Local;

@Local
public interface XASessionBeanLocal {
```

```
   String execute(String SQL,String imsusr,String imspw,
                  String SQL2,String db2usr,String db2pw);
}
```

The XASessionBeanLocal is the Interface for the EJB and defines its functions. In this case, the only function is execute() and expects the SQL commands, the user IDs, and passwords for IMS and DB2 as parameters.

XASessionBean implementation

You must implement the XASessionBean itself, as shown in Example 7-3.

Example 7-3 XASessionBean.java

```
package samples.ims;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import javax.annotation.Resource;
import javax.annotation.Resource.AuthenticationType;
import javax.ejb.Stateless;
import javax.sql.DataSource;

@Stateless //1
public class XASessionBean implements XASessionBeanLocal {
   public XASessionBean() {}

@Resource(name="imsjndixa",authenticationType=AuthenticationType.APPLICATION,shareable=true
,type=javax.sql.DataSource.class,mappedName="imsjndixa") private DataSource imsDS; //2
@Resource(name="db2jndixa",authenticationType=AuthenticationType.APPLICATION,shareable=true
,type=javax.sql.DataSource.class, mappedName="db2jndixa") private DataSource db2DS;
   public String execute(String SQL, String imsusr, String imspw,
                         String SQL2, String db2usr, String db2pw) {
      String result=""; // 3
      try{
         if(!(SQL.isEmpty())){ //4
            Connection conn = imsDS.getConnection(imsusr,imspw); //5
            Statement st =conn.createStatement();
            result+="<h3>IMS Results</h3><br>for "+SQL+"<br><br>";
            if(st.execute(SQL)){ //6
               ResultSet rs = st.getResultSet(); //7
               ResultSetMetaData rsmd = rs.getMetaData();
               int numColumns = rsmd.getColumnCount();
               result+="<table><tr>";
               for (int i=1; i<numColumns+1; i++) {
                  result += "<td>"+rsmd.getTableName(i)+".";
                  result += rsmd.getColumnName(i)+"</td>";
               }
               result+="</tr>";
               while (rs.next()) {
                  result+="<tr>";
                  for (int i=1; i<numColumns+1; i++) {
                     result += "<td>"+rs.getString(i)+"</td>"; //8
                  }
                  result+="</tr>";
               }
               result+="</table>";
               rs.close(); //9
            }else{
```

```
                result+=st.getUpdateCount() + "Rows Updated <br>"; //10
            }
            st.close(); //11
            conn.close();
        }
    }catch(Exception ex){ //12
        ex.printStackTrace();
        result+="<b>"+ex.getMessage()+"</b><br>";
    }
    try{
        if(!(SQL2.isEmpty())){ //4
            Connection conn2 = db2DS.getConnection(db2usr,db2pw); //5
            Statement st2 =conn2.createStatement();
            result+="<h3>DB2 Results</h3><br>for "+SQL2+"<br><br>";
            if(st2.execute(SQL2)){ //6
                ResultSet rs2 = st2.getResultSet(); //7
                ResultSetMetaData rsmd2 = rs2.getMetaData();
                int numColumns2 = rsmd2.getColumnCount();
                result+="<table><tr>";
                for (int i=1; i<numColumns2+1; i++) {
                    result += "<td>"+rsmd2.getTableName(i)+".";
                    result += rsmd2.getColumnName(i)+"</td>";
                }
                result+="</tr>";
                while (rs2.next()) {
                    result+="<tr>";
                    for (int i=1; i<numColumns2+1; i++) {
                        result += "<td>"+rs2.getString(i)+"</td>"; //8
                    }
                    result+="</tr>";
                }
                result+="</table>";
                rs2.close(); //9
            }else{
                result+=st2.getUpdateCount()+" Rows Updated"; //10
            }
            st2.close();
            conn2.close(); //11
        }
    }catch(Exception ex){ //12
        ex.printStackTrace();
        result+="<b>"+ex.getMessage()+"</b>";
    }
    result+="<br>Call Completed";
    return result;
  }
}
```

Here are the explanations for the commented numbers in the code in Example 7-3 on page 167:

1. This is a EJB 3.0 and uses annotations for certain declarations. This annotation defines this as a Stateless EJB, which means that it does not save any information in it.
2. These are the two Resource annotations for the IMS DataSource and the DB2 DataSource. Each annotation specifies the JNDI name, whether the Connection is shareable and the authentication type. In this case, it is AuthenticationType.APPLICATION because the application specifies the user ID and the password. The alternative is AuthenticationType.CONTAINER, which uses a Java Authentication and Authorization Service alias in WebSphere for authentication.

3. In this example, the return value is a String, which is the result String. Every Output is appended to the string. This application mixes the business logic layer with the presentation layer, which you do not do in reality. The alternative is to generate an object that holds the data in it and complete the formatting in the presentation layer.

   The DB2 code is very similar to the IMS code because it also uses the javax.sql.DataSource interface, supports the same functions, and uses the same programming approach. Therefore the numbers 4-12 appear twice in the code.

4. You can also use the application if you do not have access to IMS or DB2. This check verifies if the SQL String is empty. If it is empty, it does not open a connection to the resource.
5. This statement gets a Connection through JNDI referenced javax.sql.DataSource Object by specifying the user ID and password for authentication on the target system.
6. With this command the Statement executes the SQL. The return type indicates if your SQL is manipulating or just querying data. If it is a query and the return value is true, it goes in the if branch; otherwise, it goes in the else branch.
7. If you branch to the query, you get to the ResultSet with the st.getResultSet() function. Then it generates the header in the format <tablename.columnname> and puts the results in an HTML table.
8. This scenario assumes that all return types are Strings. If not, it tries to cast the result as a String. If this does not work, it fails with an exception.
9. After reading the results, the ResultSet Object can be closed and the data can be released.
10. In the manipulate data branch, if you do an INSERT, UPDATE, or DELETE, you get the number of affected rows back as a result.
11. After finishing all work, the Statement and Connection Objects can be marked as ready for closing, which does not mean that the Connection is really closed because in a Global transaction, the Container decides when to close the Connection object. Normally this occurs after committing or rolling back all changes in the resources that are involved in the two phase-commit process.
12. By calling the Connection, several methods can throw Java Exceptions, for example, if the user authentication fails. These exceptions are caught and appended to the result to make them visible on the result web page.

XAServlet implementation

Example 7-4 shows the sample code for the files in the web project.

Example 7-4 XAServlet.java

```
package samples.imsservlet;
import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import samples.ims.*;

public class XAServlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

   public XAServlet() {
```

```
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        doPost(request,response);
    }

    @EJB(beanName="XASessionBean") private XASessionBeanLocal bean;
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
            String result=" ";
            HttpSession session = request.getSession(true);
            String query =request.getParameter("query").trim();
            String imsusr=request.getParameter("imsusr").trim();
            String imspw =request.getParameter("imspw").trim();
            String query2=request.getParameter("query2").trim();
            String db2usr=request.getParameter("db2usr").trim();
            String db2pw =request.getParameter("db2pw").trim();
            result = bean.execute(query,imsusr,imspw,query2,db2usr,db2pw);
            session.setAttribute("result", result);
            RequestDispatcher disp=getServletContext().getRequestDispatcher("/XATest.jsp");
            disp.forward(request, response);
    }
}
```

The coding of the XAServlet is kept very simple. It uses the usual methods with doGet and doPost of a HttpServlet. The doGet method forwards the request to the doPost method. It catches the parameters from the web site form element and passes it to the EJB.execute() method. The response result is placed in the session object. The RequestDispatcher forwards the response to the jsp web site again.

XATest JSP web site

Example 7-5 shows the source code of the XATest.jsp web site.

Example 7-5 XATest.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
   language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<html><head>
<title>IMS and DB2 Access via JDBC XA</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head><body>
<h1>IMS and DB2 Access via JDBC XA DataSource</h1>
<form method="post" action="/IMSandDB2Web/XAServlet">
<table>
<tr><td><b>IMS</b></td></tr>
<tr>
<td><input type="text" name="query" value="SELECT * FROM AUTOLPCB.DEALER" size="50"></td>
<td>User</td><td><input type="text" name="imsusr" size="9"></td>
<td>PW</td><td><input type="password" name="imspw" ></td>
</tr>
<tr><td><b>DB2</b></td></tr>
<tr>
<td><input type="text" name="query2" value="SELECT * FROM DSN8910.EMP" size="50"></td>
<td>User</td><td><input type="text" name="db2usr" size="9"></td>
<td>PW</td><td><input type="password" name="db2pw"></td>
</tr>
```

```
<tr><td><input type="submit" name="sub" value="Execute"></td></tr>
</table>
</form>
<BR>
<%=(String)session.getAttribute("result") %>
</body>
</html>
```

The XATest.jsp file contains only usual HTML elements. When **Submit** is clicked, it forwards the values to the XAServlet, and after getting the result, it reads the result String from the session context.

7.2.5 Exporting the application

Now the application is ready for deployment in WebSphere Application Server. Rational Application Developer offers you an easy way to publish your application to a WebSphere Application Server, but very often you have restrictions in accessing the administration port directly. For this reason, in this scenario, we show how that must always work to export the application and install it through the administration web console of the WebSphere Application Server:

1. Right-click your EAR Project, and select **Export** → **EAR file**
2. Choose a destination, such as, c:\temp\DB2andIMS.ear, and click **Finish**.

The EAR project automatically contains all referenced Projects, such as the web project and the EJB project and are part of the EAR file.

7.2.6 Setting up the IMS Universal DB Resource Adapters in WebSphere Application Server 7.0

You must install the correct IMS Universal DB Resource Adapter and configure it afterwards, which you can do with the help of the WebSphere Application Server administrative console. When you have WebSphere Application Server running, you can access the administrative console in your web browser. The default URL is:

`http://localhost:9061/ibm/console`

Whether you have security enabled or not, you must enter your Used ID and Password, or leave them blank to login.

Installing the IMS Universal DB Resource Adapter

To install the IMS Universal DB Resource Adapter:

1. When the WebSphere configuration web site is open, expand **Resources** → **Resource Adapters** → **Resource Adapters**.
2. Select the scope where you want to install the Resource Adapter. Normally you deploy it to the whole node, as shown in Figure on page 171.

Figure 7-10 WebSphere Application Server Console

3. Click **Install RAR**. Specify the location of the IMS Universal DB Resource Adapter on your local file system, and click **Next**. In this scenario, we use the imsudbJXA.rar Resource Adapter because the application uses JDBC and requires Global XA Transaction support.
4. Specify the class path. The Class path must contain any additional resources that are needed. In the case of IMS, the IMS Universal Resource Adapters need the Metadata class files to access an IMS database.
5. The Metadata Class files can be part of the Class path of the IMS Universal DB Resource Adapter or can be part of the application. To be more flexible, it is easier to specify it in the class path of the Resource Adapter. That way, if you change your database layout, you do not have to rebuild all applications which contain the metadata file.
6. In this case we are specifying it as part of the Resource Adapter Class path. Therefore refer the class path to the jar file with the Metadata class file in it for the Car Dealer IVP Database. In this example, the path points to c:\temp\AUTPSB11.jar, as shown in Figure 7-11 on page 173. If you specify more than one jar file, do not use separators; instead, use a new line.
7. Click **OK** and save the changes to the Master configuration.

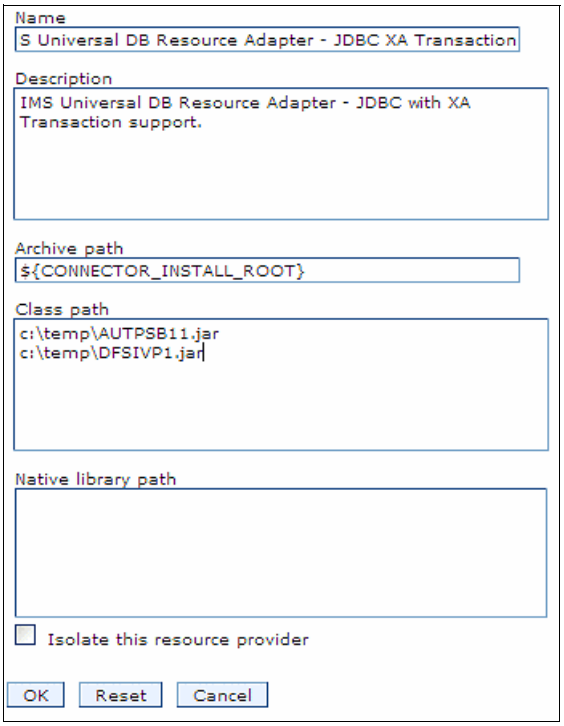


Figure 7-11 WebSphere Application Server Console: Specify Class path

Creating J2C Connection factories

After installing the driver, you must make it usable by the application by specifying a connection to a system, which you do by creating a J2C Connection factory:

1. In the administration console, click **Resources** → **Resource Adapters** → **J2C connection factories.** Select the same scope that you used when you installed the Resource Adapter, and click **New**.
2. Select the installed Resource Adapter from the Provider list, and specify the name and JNDI name as **imsjndixa**, as shown in Figure 7-12 on page 174, and click **Apply**.

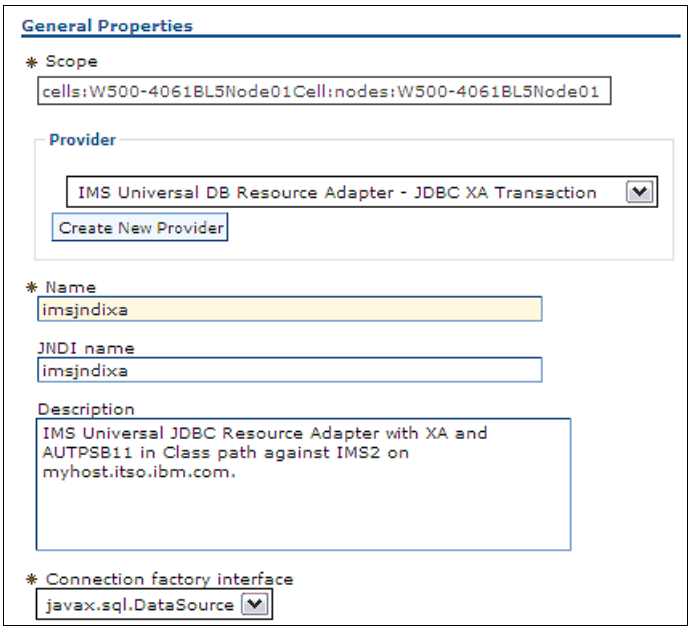


Figure 7-12 WebSphere Application Server Console: New J2C connection factory

3. After clicking **Apply**, you can click the **custom properties** link, and edit the values accordingly to your system settings:
   - The SSLConnection parameter indicates if SSL encryption is enabled for the connection to IMS Connect or not. Set this value to **false** because this scenario does not use SSL.
   - The Data storeName is the name of the target IMS data store. If you specified an alias name in your ODBM configuration member on the mainframe side, you must use the alias name instead of the IMSID. In this example, this value is IMS2.
   - The Data storeServer is the name or IP address of the target IMS Connect. In this example, the value is myhost.itso.ibm.com.
   - The PortNumber specifies the port number of the target IMS Connect. In this example, the value is 5555.
   - The DriverType specifies the connectivity type. In the case of the XA Resource Adapter, this value must be set to 4.
   - The LoginTimeout specifies the number of seconds to wait for a TCP/IP response from IMS Connect. A value of 0 indicates an unlimited wait time. Change this value to 10, so that your application does not wait forever in case of an error.
   - The MetadataURL is the URL of the database metadata that represents the target IMS database. It is built by the pattern class://package.PSBDatabaseView. In our case, the value is class://samples.ims.openDb.AUTPSB11DatabaseView.
   - The User and Password specifies the user ID and the password for the default user. This user is used if you do not specify a user explicitly and do not provide a JAAS alias to the Resource Adapter. Because we specify these values explicitly in the application by invoking the connection, you do not have to specify them, but you can if you want to.
4. After you finish editing your settings, click **save your changes to the master configuration.**

7.2.7 Setting up IBM (DB2) Data Server Driver for JDBC and SQLJ in WebSphere Application Server 7.0

The installation of the DB2 Data Server Driver is similar to the installation of the IMS Universal Driver. The difference is that we are not using a DB2 Resource Adapter; instead, we use the Driver directly as a JDBC Provider. If you want to know more about the IBM Data Server Drivers, refer to Appendix A.1, “IBM Data Server drivers and clients” on page 230.

Installing the JDBC Provider

To install the JDBC Provider, you must first specify the Driver location, which is made available for use in a specific scope:

1. Click **Resources** → **JDBC** → **JDBC Providers**. Select the Scope for the installation, and click **New.** Select **DB2** as the Database type, and select the **JCC Driver** with the **XA data source** implementation, as shown in Figure 7-13. Click **Next**.

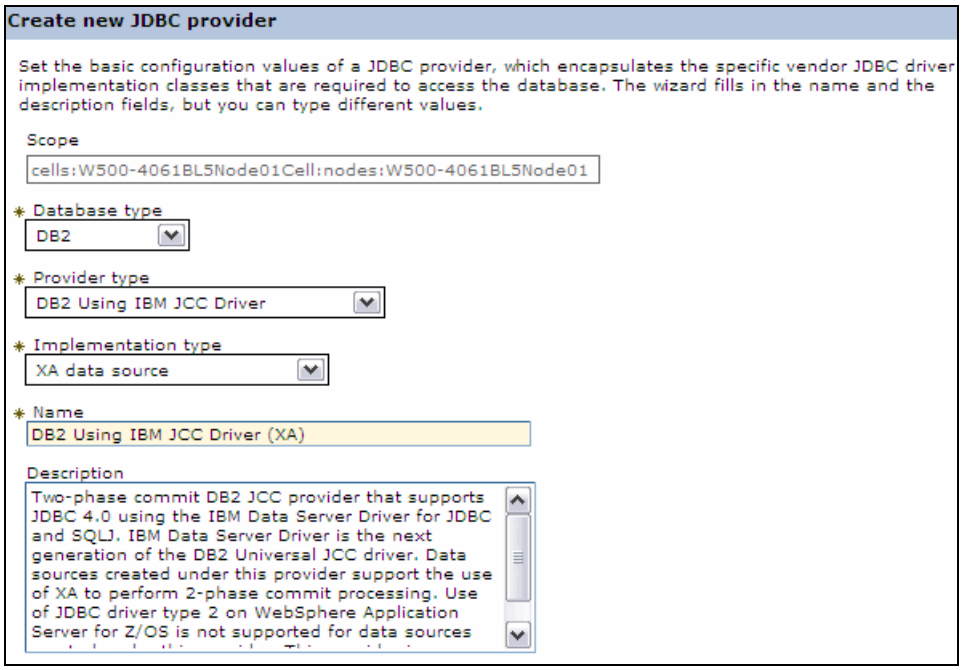


Figure 7-13 WebSphere Application Server Console: New JDBC Provider

2. Specify the Class path where your DB2 Data Server Driver jar files are located, for example, specifying `c:\temp\db2` ensures that WebSphere automatically searches for db2jcc4.jar, db2jcc_license_cu.jar and db2jcc_license_cisuz.jar in that directory. In the Summary page, click **Next** → **Finish** → **save the changes to the master configuration**.

Setting up the DataSource

Next, you must create a DataSource for a specific Database connection that applications with its JNDI name can reference:

1. Click **Resources** → **JDBC-** → **Data sources.** Select the same scope as for the Driver installation and click **New**. Specify **db2jndixa** as name and JNDI name and Click **Next.**
2. Select an **existing JDBC Provider** and choose **DB2 Using IBM JCC Driver (XA)** and click **Next**.

Specify the following attributes:

- Driver type as 4.
- Database name as your Sub System ID for your DB2 on System z database (for example, DB9A).
- Server name, which is the host or IP address of the target DB2 (for example, myhost.itso.ibm.com).
- Port number (default is 50000).

3. On the Summary page, click **Next** → **Next** → **Finish** → **Save the changes to the master configuration**.

7.2.8 Installing and starting the application

After setting up the required DataSources for IMS and DB2, you can deploy the application in WebSphere Application Server:

1. In the WebSphere Application Server administration console, expand **Applications** → **New Application**. Click **New Enterprise Applications**.
2. Select the exported ear file from your local file system, and click **Next**.
3. Select the **Fast Path** option for a quick installation, and click **Next**.
4. Leave the defaults on the following pages, and click **Next** → **Next** until you come to the Map Resource References.
5. Click **Browse** for each DataSource, and select the matching JNDI DataSource, as shown in Figure 7-14.

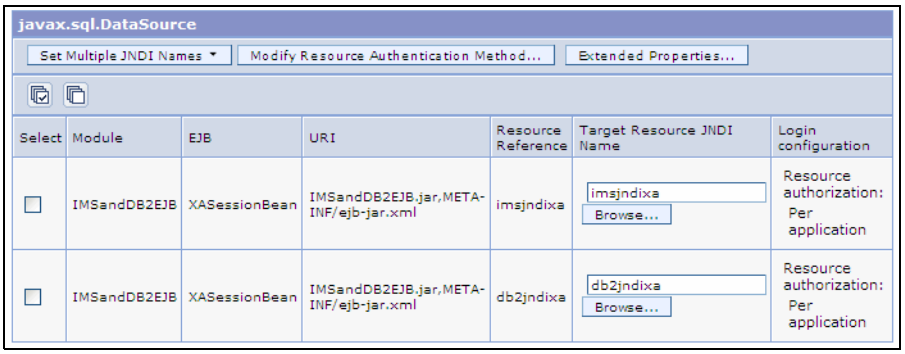


Figure 7-14 WebSphere Application Server Console: JNDI Mapping

6. Click **Next** → **Finish**. Ensure that the application is installed successfully. Click **Save the changes to the master configuration**.

   Now the application is installed but not yet started. To start the application:

   a. Click **Applications** → **Application Types** → **WebSphere enterprise applications**.
   b. Select your application, and click **Start**. Check that the status icon turns to green.

7.2.9 Running the application

Now the application works. Open a web browser, and reach your application through the following link:

`http://localhost:9081/IMSandDB2Web/XATest.jsp`

The IP address and port number might be different. The IP address is the IP address or host name of the server that is running WebSphere Application Server. The port number can vary because of the settings on the Application Server. To determine which port number to use:

1. From the WebSphere administrative console, click **Servers** → **Server Types** → **WebSphere Application servers**.
2. Under the Communications heading on the right side, click your server, and see what the value is for WC defaulthost. In this example it is 9081.

The web page you will see is very simple. It contains fields for SQL commands against IMS, DB2, and for the user ID and password for each connection. You can play around with issuing SQL commands. If you want, you can use just one connection and leave the SQL for the other connection empty.

Figure 7-15 shows the result of a query against IMS and DB2.

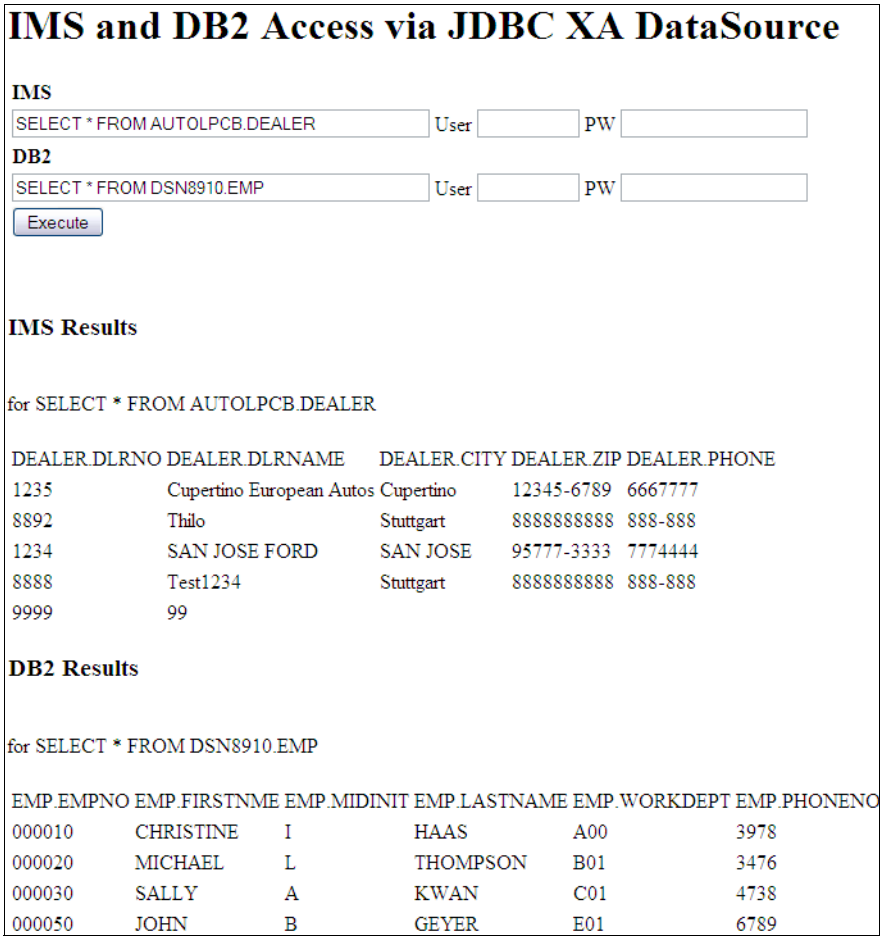


Figure 7-15 Managed Application web site results

7.3 Developing an IMS Java Transaction using the IMS Universal JDBC driver

IMS offers two types of IMS dependent regions for Java, the JMPs and JBPs:

- JMPs are Java Message Processing regions that can run IMS transactions that are written in Java.
- JBPs are Java Batch Processing regions that can run IMS batch jobs that are written in Java.

To write an application that can receive messages and reply messages, you must write an application for JMP regions.

IMS Version 11 introduces a new IMS Java-dependent region resource adapter that provides an enhanced programming API for writing JMP and JBP applications for IMS. We use this API in this scenario.

For more information about writing with this adapter, refer to *IMS Version 11 Application Programming*, SC19-2428 and *IMS Version 11 Application Programming APIs*, SC19-2429.

This scenario shows an IMS Java Transaction that reads the SQL Code in the input message and returns the result as an output message. The input and output message classes extend the com.ibm.ims.application.IMSFieldMessage.

Example 7-6 shows the code for the InputMessage.

Example 7-6 InputMessage.java

```
package samples.ims.openDb;
import com.ibm.ims.base.*;
import com.ibm.ims.application.IMSFieldMessage;

public class InputMessage extends IMSFieldMessage {

    static DLITypeInfo[] fieldInfo = {
        new DLITypeInfo("SQL",     DLITypeInfo.CHAR,  1, 80),
    };

    public InputMessage() {
        super(fieldInfo,89, false);
    }
}
```

The InputMessage defines only one DLIType with the length of 80 Bytes, which is for the SQL command.

Example 7-7 shows the OutputMessage. It defines only one field with the length of 220 Bytes for the result of the transaction.

Example 7-7 OutputMessage.java

```
package samples.ims.openDb;
import com.ibm.ims.base.*;
import com.ibm.ims.application.IMSFieldMessage;

public class OutputMessage extends IMSFieldMessage {

    static DLITypeInfo[] fieldInfo = {
```

```
        new DLITypeInfo("Results",DLITypeInfo.CHAR,1,220),
    };

    public OutputMessage() {
        super(fieldInfo,220, false);
    }
}
```

The transaction program with the main method is the CarDealerTrans class. This class uses the IMS Java dependent region resource adapter, as shown in Example 7-8.

Example 7-8 CarDealerTrans

```
package samples.ims.openDb;
import com.ibm.ims.dli.DLIException;
import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
import com.ibm.ims.dli.tm.IOMessage;
import com.ibm.ims.dli.tm.MessageQueue;

public class CarDealerTrans {
    public static void main(String[] args) throws DLIException {
        Application app = ApplicationFactory.createApplication();
        MessageQueue mq = app.getMessageQueue();
        IOMessage inMsg = app.getIOMessage("class://samples.ims.openDb.InputMessage");
        IOMessage outMsg = app.getIOMessage("class://samples.ims.openDb.OutputMessage");

        CarDealerDBInteractions dbcalls = new CarDealerDBInteractions();
        String host = "myhost.itso.ibm.com";
        int port = 5555;
        String datastoreName = "IMS2";
        String username = "USRID";
        String password = "PASS";

        while (mq.getUnique(inMsg)) {
            String sql = inMsg.getString("SQL");
            String output = dbcalls.sqlMethod(sql,4,host,port,
                              datastoreName,username,password);
            outMsg.setString("Results", output);
        }
    }
}
```

The main application accesses the message queue and retrieves messages from it, for as long as there are messages for the application on the queue. It defines a CarDealerDBInteractions object and calls the sqlMethod() by passing the required parameters to it. The application can easily switch between type-2 and type-4 connectivity, which is helpful if you want to access another database in a remote IMS. You can easily access two databases in two different IMS systems. Several scenarios are possible.

The database calls are separated in the CarDealerDBInteraction class. This class uses the IMS Universal JDBC driver. See Example 7-9.

Example 7-9 CarDealerDBInteraction

```
package samples.ims.openDb;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import com.ibm.ims.jdbc.IMSDataSource;

public class CarDealerDBInteractions {
   public String sqlMethod(String sqlQuery, int driverType, String host,int port,
            String datastoreName, String username,String password){
      String result = "";
      IMSDataSource ds = new IMSDataSource();
      ds.setMetadataURL( "class://samples.ims.openDb.AUTPSB11DatabaseView" );
      ds.setDatastoreName(datastoreName);
      ds.setDriverType(driverType);

      if(driverType==4){
         ds.setDatastoreServer(host);
         ds.setPortNumber(port);
         ds.setUser(username);
         ds.setPassword(password);
      }
      try {
         Connection conn = ds.getConnection();
         Statement st = conn.createStatement();
         ResultSet rs = st.executeQuery(sqlQuery);
         while(rs.next()){
            for(int i=1; i<=rs.getMetaData().getColumnCount();i++){
               result += rs.getString(i) + "\t";
            }
            result += "\n";
         }
         if(result == ""){
            result += "No records match the provided query";
         }
         rs.close();
         st.close();
         conn.commit();
         conn.close();
      } catch (SQLException e) {
         e.printStackTrace();
         result = result + e.toString() + "\n";
      }
      return result;
   }
}
```

This class defines an IMSDataSource object and specifies the required parameters, depending on type-2 or type-4 connectivity. Because the application only allows queries against the database, the sample does not need a commit on the Connection.

To set up the transactions, you must follow the usual steps that are required when adding a new IMS Java transaction. The JMP region must be configured correctly for the use of the IMS Universal Drivers in IMS Java transactions. The region startup procedure contains the ENVIRON and JVMOPMAS parameters that specify the members in the IMS PROCLIB. These configuration files must contain the IMS Universal JDBC driver jar file in the class path and also the metadata class file. In our example, this is the generated AUTPSB11.jar.

If you use type-2 connectivity, you also must specify the location of the native Drivers for type-2 connectivity.

8

Scenario 3: Writing DL/I and mixed applications

In this chapter, we show how you can access your IMS databases using the IMS Universal DL/I driver and the IMS Universal DB Resource adapter with the CCI programming approach. This chapter shows how you can use the IMS Universal DL/I driver to write stand-alone DL/I applications in Java and optionally how to use batch methods. It also shows how to use the IMS Universal DB Resource Adapter with the CCI programming model to write applications using mixed SQL and DL/I syntax in managed or non-managed environments.

In this chapter, the topics that we discuss are:

- Writing applications with the IMS Universal DL/I driver
- Writing applications with the IMS Universal DB Resource Adapter and the CCI programming approach

8.1 Writing applications with the IMS Universal DL/I driver

You can use the IMS Universal DL/I driver for writing applications in Java using the DL/I syntax to access IMS databases. As with all of the IMS Universal drivers, they can be used by remote applications that use the type-4 connectivity and by local applications that use type-2 connectivity. We also show how you can batch up your accesses where appropriate to improve the performance when using the IMS Universal DL/I driver.

8.1.1 Accessing IMS data with the IMS Universal DL/I driver

To execute DL/I calls from your IMS Universal driver application, you must connect to an IMS database, which you do using the PSB interface that is part of the com.ims.dli package. In this section, we discuss what needs to be provided to the interface and provide an example of the code.

Connecting to an IMS database

The connection properties are passed using an IMSConnectionSpec instance that is created by calling the createIMSConnectionSpec method in the IMSConnectionSpecFactory class. You must set the following connection properties for the IMSConnectionSpec instance:

- data storeName as the name of the data store to be accessed:
  - For type-4 (remote) connectivity, the data store name must either match the data store name that is defined in the active ODBM CSLDCxxx proclib member or be blank. If the data store name is defined using the `ALIAS(NAME=` parameter, you must use the name that is specified as the alias. If the data store name is left blank or not supplied, IMS connects to any available ODBM because it assumes that data sharing is enabled among all data stores that are defined to ODBM.
  - For type-2 (local) connectivity, the data store name is the name of the IMS data store to access. Set the data storeName property to the IMS subsystem alias name. In an IMS Java Dependent Region (JDR) runtime there is no DFSPRP member required.
- metadataURL is the fully qualified name of the JAVA metadata class that the IMS Enterprise Suite DLIModel utility plug-in generates. The URL must be prefixed with class://. In the application example, it is defined as: class://samples.ims.openDb.AUTPSB11DatabaseView
- portNumber is the Port Number of IMS Connect:
  - For type-4 (remote) connectivity, this is the TCP/IP server port number that is used to communicate with IMS Connect. It is defined using the DRDAPORT parameter of the ODACCESS statement in the integrated IMS Connect configuration proclib member.
  - For type-2 (local) connectivity, do not set this property.
- data storeServer is the IP address of the IMS Connect host:
  - For type-4 (remote) connectivity, this is the name or IP address of the data store server (IMS Connect). In the sample application, we specified the name, wtsc63.itso.ibm.com, but we can use its IP address, 9.12.6.70, instead.
  - For type-2 (local) connectivity, do not set this property.
- driverType is used to specify the type of driver connectivity the application will be using:
  - For type-4 connectivity, the value must be IMSConnectionSpec.DRIVER_TYPE_4 or 4
  - For type-2 connectivity, the value must be IMSConnectionSpec.DRIVER_TYPE_2 or 2

- sslConnection is an optional property used to indicate if Secure Socket Layer (SSL) is being used for the connection:
  - For type-4 (remote) connectivity, set it to true to enable SSL or false if SSL is not used.
  - For type-2 (local), do not set the property.
- loginTimeout is an optional property that specifies the number of seconds that the driver waits for a response from the server before timing out:
  - For type-4 connectivity, set the value to a non-negative integer for the number of seconds. Setting it to 0 tells the driver to wait forever.
  - For type-2 connectivity, do not set this property.
- user is the username for authentication to IMS Connect. Do not set this property for type-2 connectivity.
- password is the password that is used for the connection. Do not set this property for type-2 connectivity.

Example 8-1 shows how to specify the values.

Example 8-1 IMSConnectionSpec properties

```
// establish a database connection
IMSConnectionSpec connSpec
= IMSConnectionSpecFactory.createIMSConnectionSpec();
connSpec.setDatastoreName("IMS2");
connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
connSpec.setPortNumber(5555);
connSpec.setMetadataURL(
      "class://samples.ims.openDb.AUTPSB11DatabaseView");
connSpec.setUser("IMS2R");
connSpec.setPassword("password");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
```

Having set your connection properties you now must pass the connection properties to the PSBFactory class to create a PSB instance. After the PSB instance is successfully created, you will have a connection to the database. See Example 8-2.

Example 8-2 Creating a PSB instance

```
psb = PSBFactory.createPSB(connSpec);
```

8.1.2 Retrieving Data using the IMS Universal DL/I drivers

In a traditional, that is COBOL, PL/1, Assembler, IMS application to delete, insert, replace or retrieve data DL/I calls are made. Using the IMS Universal DL/I driver to perform those same functions, you invoke methods.

These methods are defined in the following interfaces:

- You saw how the PSB instance is created to connect to the database and that PSB interface can be used to obtain a handle on any Program Communication Block (PCB) within the PSB. The PCB handle is used to access the particular IMS database that is referenced by that PCB.
- The PCB interface represents a cursor position in an IMS database. The PCB interface supports DL/I call functions, including Get Unique (GU), Get Next (GN), Get Next within Parent (GNP), Insert (ISRT), Replace (REPL) and Delete (DLET). The PCB interface can obtain an unqualified list of Segment Search Arguments (SSAs) and perform batch

retrieve, update and delete operations. You can also use the PCB interface to return the Application Interface Block (AIB) that is associated with the most recent DL/I call.

- The SSAList interface represents a list of SSAs that specify which segments to target in a particular DL/I call. You use the SSAList to construct the SSAs and to set any command codes and lock classes that the segments referenced. You can set an initial qualification statement and append additional qualifiers based on the values of the segment fields to restrict which segments are targeted. Each SSA in the list can be qualified or unqualified. You can also specify which fields from segments are to be returned by a retrieve call.
- The Path interface represents a database record for the purpose of a retrieval or update operation. It can be viewed as the concatenation of all of the segment instances in a specific hierarchic path, starting from the highest level segment that is nearest to the root segment to the lowest level segment. Use the path interface to set or retrieve the value of any segment field that is located in the hierarchic path.
- The PathSet interface provides access to a collection of Path objects that are returned by a batch retrieve operation.
- The AIB interface and the database PCB(DBPCB) interface return useful information that IMS returns as a result of a DL/I call.
- The GSAMPCB interface represents a GSAM PCB and is essentially a cursor position in a GSAM database. The interface provides data access to GSAM databases with calls that are similar to DL/I calls.
- The RSA interface represents a GSAM database record search argument that is the key to a cursor position in the GSAM database.

Example 8-3 shows how to get a handle on the AUTOLPCB PCB that is specified in the AUTPSB11 PSB and uses the AUTOLDB database that is specified in that PCB to show how an unqualified SSAList can be constructed. The example gets a path instance using the SSAList and calling the getPathFor RetrievalReplace method and uses the getUnique method to return a Path that consists of all fields in the STOCSALE segment.

Example 8-3 Obtaining a PCB handle and specifying SSAs using the SSAList interface

```
pcb = psb.getPCB("AUTLPCB");
SSAList ssalist = pcb.getSSAList(“DEALER”,”STOCSALE”);
Path path = ssaList.getPathForRetrievalReplace();
pcb.getUnique(path, ssalist,false);
```

In Example 8-3, the SSAList represents all segments along the hierarchic path from the topmost segment, DEALER, to the lowest segment, STOCSALE. The SSAList will look like:

```
DEALERbbb
MODELbbbb
STOCKbbbb
STOCSALEb
```

An SSAList can be qualified to filter the segments in the hierarchic path to be retrieved or updated. Generally the steps to create a qualified SSAList are:

1. Use the getSSAlist method to create an unqualified SSAList from the PCB.
2. Use the addInitialQualification method to specify the initial search criteria for a segment in the SSAList returned by a getSSAlist method. You can use one addInitialQualification statement for each segment represented in the SSAList. If you use more than one addInitialQualifaction statement for a segment, an exception error is thrown. The relational operator parameter in the addInitialQualification method indicates the conditional criteria that the segment must meet to be qualified.

The valid operators are:

```
EQUALS
GREATER_OR_EQUAL
GREATOR_THAN
LESS_OR_EQUAL
LESS_THAN
NOT_EQUAL
```

3. To add additional search criteria to a segment in the SSAList, you must use the appendQualification method. The Boolean Operator (BooleanOp) parameter in the appendQualification method statement says how this qualification is logically connected to the previous qualification. The valid BooleanOp values are:

```
AND
OR
INDEPENDENT_AND
```

4. Segment SSAs in the SSAList can also be qualified by setting DL/I command codes and lock classes. Supported DL/I command codes are:

```
CC_C: The C command code (Concatenated Key). Use the addConcatenatedKey method
       to add the concatenated key to the segments SSA in the SSAList.
CC_D: The D command code (path call).
CC_F: The F command code (first occurrence).
CC_L: The L command code (last occurrence).
CC_N: The N command code (path call ignore).
CC_P: The P command code (set parentage).
CC_U: The U command code (maintain position at this level).
CC_V: The V command code (maintain position at this and all higher levels)
```

The lock class prevents another application from updating a segment until your program reaches a commit point. Use the addLockClass method to add a lock class to a segment. The supported lock class letters are A to J. The behavior of a lock class is the same as using a Q command code with the lock class letter.

Example 8-4 shows how to specify and use a qualified SSAList with just an initial qualification statement to retrieve data.

Example 8-4 Qualified SSAList with initial qualification

```
SSAList ssalist = pcb.getSSAList(“DEALER”,”STOCSALE”);
ssalist.addInitialQualification(“MODEL”,”MAKE”,SSAList.EQUALS,”FORD”);
ssalist.markFieldForRetrieval(“STOCK”,”COLOR”,true);
Path path = ssaList.getPathForRetieveReplace();
pcb.getUnique(path, ssaList, false);
```

The SSAList for Example 8-4 is:

```
DEALERbbb
MODELbbb(MAKEbbbbEQFORDbbbbbb)
STOCKbbb*D
STOCSALEb
```

The data returned by Example 8-4, where MAKE =Ford is the COLOR field in the STOCK segment and all the fields of segment STOCSALE because by default, IMS returns all fields of the lowest level segment specified.

Example 8-5 shows how we can add an extra qualification to the SSAList for the MODEL SSA in Example 8-4 on page 185.

Example 8-5 Qualified SSAList with multiple qualifications for one SSA

```
SSAList ssalist = pcb.getSSAList(“DEALER”,”STOCSALE”);
ssaList.addInitialQualification(“MODEL”,”MAKE”,SSAList.EQUALS,”FORD”);
ssaList.appendQualification(“MODEL”,SSAList.AND,”YEAR”,SSAList.Greater_than,
2002);
ssaList.marFieldForRetrieval(“STOCK”,”COLOR”,true);
.
.
```

Example 8-6 shows how we code a qualified SSAList with the command code CC_P (set parent). DL/I, by default, sets parent at the lowest level in the path of segments returned from the SSAs specified in the SSAList when a GU or GN call is issued.

Example 8-6 Qualified SSA using a command code

```
SSAList ssalist = pcb.getSSAList(“DEALER”,”STOCSALE”);
ssaList.addInitialQualification(“MODEL”,”MAKE”,SSAList.EQUALS,”FORD”);
ssaList.addCommandCode(“MODEL”,SSAList.CC_P);
ssaList.appendQualification(“MODEL”,SSAList.AND,”YEAR”,SSAList.Greater_than,
2002);
ssaList.marFieldForRetrieval(“STOCK”,”COLOR”,true);
.
.
```

The SSAList for our example is:

```
DEALERbbb
MODELbbb*P(MAKEbbbbEQFORDbbbbbb&YEARbbbbGT2002)
STOCKbbbb
STOCSALEb
```

The IMS Universal DL/I driver provides support for data retrieval that mirrors DL/I semantics.

We showed you how to:

- Obtain an SSAList instance from the PCB instance representing the database.
- Add qualification statements to the SSAList.
- Specify the segment fields to retrieve and get a Path instance using the SSAList instance and calling the getPathForRetrieveReplace method, which causes all of the fields to be retrieved in the resulting Path object that were marked for retrieval.

We also mentioned the getUnique method and this and other DL/I retrieve methods are shown in Table 8-1 on page 187. There is one more thing to say about marking fields for retrieval, and that is that by default all of the fields in the lowest level segment specified in the SSAList are returned. However, if any individual fields in that segment are marked for retrieval only those marked fields are returned.

Table 8-1 Methods for DL/I retrieve from the PBC interface

JAVA API for DL/I retrieve method	Use
getUnique	Retrieves a specific unique segment. Method provides the same functionality as a DL/I GU database call. If the isHoldCall parameter is set to true, the method behaves as a DL/I GHU database call.
getNext	Retrieves the next segment in the path. Method provides the same functionality as a DL/I GN database call. If the isHoldCall parameter is set to true, the method behaves as the DL/I GHN database call does.
getNextWithinParent	Retrieves the next segment within the same parent. Method provides the same functionality as the DL/I GNP database call. If the isHoldCall parameter is set to true, the method behaves as a DL/I GHNP database call does.
batchRetrieval	Retrieves multiple segments with a single call.

Methods for retrieving and converting data types

Data can be retrieved and converted from how it is defined in the database metadata to the type your Java application is expecting by using the Path interface. If the application was using either the IMS Classic JDBC or IMS Universal JDBC drivers, you can use the ResultSet interface.

Table 8-2 shows the get methods that you can use in the ResultSet interface or the Path interface for accessing certain data of a certain Java data type.

Table 8-2 The get methods

ResultSet.getXXX or Path.getXXXX Method	No truncation or data loss	Legal without data integrity
getByte	TINYINT	SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getShort	SMALLINT	TINYINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getInt	INTEGER	TINYINT, SMALLINT, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getLong	BIGINT	TINYINT, SMALLINT, INTEGER, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getFloat	FLOAT	TINYINT, SMALLINT, INTEGER, BIGINT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL1, ZONEDDECIMAL1
getDouble	DOUBLE	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getBoolean	BIT	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getString	CHAR VARCHAR	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, PACKEDDECIMAL, ZONEDDECIMAL, BINARY, DATE, TIME, TIMESTAMP

ResultSet.getXXX or Path.getXXXX Method	No truncation or data loss	Legal without data integrity
getBigDecimal	PACKEDDECIMAL ZONEDDECIMAL	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR
getClob	CLOB	all others result in an exception
getBytes	BINARY	all others result in an exception
getDate	DATE	CHAR, VARCHAR, TIMESTAMP
getTime	TIME	CHAR, VARCHAR, TIMESTAMP
getTimestamp	TIMESTAMP	CHAR, VARCHAR, DATE, TIME

Note that:

- PACKEDDECIMAL and ZONEDDECIMAL are data type extensions for the IMS Classic JDBC, IMS Universal JDBC, and IMS Universal DL/I drivers. All other types are standard SQL types that are defined in SQL92. PACKEDDECIMAL and ZONEDDECIMAL do not support the Sign Leading or Sign Separate modes. For these two data types, data is always stored with the Sign Trailing method.
- The CLOB data type is only supported for the storage and retrieval of XML, which is currently only supported by the IMS Classic JDBC driver.

8.1.3 Inserting data using the IMS Universal DL/I driver

There are two methods, the create and the insert, that can be used in the PCB interface to add a new segment into the database. Used with the IMS Universal DL/I driver the two methods provide functionality that is similar to the DL/I ISRT call. The insert method returns an IMS status code that indicates the result of the operation, and the create method returns a count of the number of segments inserted (this currently will always return 1). The two methods provide defaults for any DBD search fields that are not explicitly set:

- Character-based fields are set to blanks.
- Numeric based fields are set to zero.
- Non-DBD fields are set to hex ‘00’.
- If a key field is not set, an exception is thrown.
- If a key field is broken down into multiple sub fields in the DBD, you must use the key field to set the value.
- If just the sub fields are allocated values, an exception is thrown.

You can specify values for both, but make sure they line up identically. If you provide values for both the key field and the sub fields and the values are not the same, the actual value of the key field for the segment that is inserted into the database depends on the order of the path.setString statements in the application. Last overrides first.

Example 8-7 shows how to use the create and insert methods to add a new MODEL and two ORDER segments into the AUTODB where the DLRNO is 1234.

Example 8-7 Create and Insert method

```
SSAList ssalist = pcb.getSSAList(“DEALER”,”ORDER”);
ssaList.addInitialQualification(“DEALER”,”DLRNO”,SSAList.EQUALS,”1234”);
Path path = ssaList.getPathForInsert(“MODEL”);
```

```
path.setString(“MODEL”, “MODKEY”, “Lotus     Evora     2010”);
path.setString(“MODEL”, “MSRP”, “65000”)
path.setString(“ORDER”, “ORDNBR”, “258921”);
path.setString(“ORDER”, “LASTNME”, “Thilo”);
path.setString(“ORDER”, “FIRSTNME”, “LIEDLOFF”);
path.setString(“ORDER”, “DATE”, “04-10-2010”);
int i = pcb.create(path, ssaList); // returns i = 1 if successful

SSAList ssalist = pcb.getSSAList(“ORDER”);
path = ssaList.getPathForInsert(“ORDER”);
path.setString(“ORDNBR”, “258930”);
path.setString(“LASTNME”, “Angelique”);
path.setString(“FIRSTNME”, “GREENHAW”);
path.setString(“DATE”, “04-09-2010”);
short status = pcb.insert(path,ssaList); // returns an IMS status code
```

8.1.4 Updating data with the IMS Universal DL/I driver

In this section, we show you how to update data with the IMS Universal DL/I driver.

To persist changes made to the database, your application must call the PSB commit method prior to deallocating the PSB. Failure to do so results in changes being rolled back to the last point that commit was called or the start of the application, if commit was never called

The replace method amends data in a segment that exists in the database.

Example 8-8 shows how the ORDER segment that might have been inserted in the previous example has the LASTNME and FIRSTNME fields corrected because the values were inserted the wrong way.

Example 8-8 The replace method

```
SSAList ssalist = pcb.getSSAList(“DEALER”,”ORDER”);
ssaList.addInitialQualification(“DEALER”,”DLRNO”,SSAList.EQUALS,”6788”);
ssaList.addInitialQualification(“MODEL”,”MODKEY”,
SSAList.EQUALS,”Lotus     Evora     2010”);
ssaList.addInitialQualification(“ORDER”,”ORDNBR”,SSAList.EQUALS,”258921”);
Path path = ssaList.getPathForRetrieveReplace();
if(pcb.getUnique(path, ssaList,true)) {
   path.setString(“LASTNME”, “LIEDLOFF”);
   path.setString(“FIRSTNME”, “Thilo”);
   pcb.replace(path);
}
```

8.1.5 Deleting data with the IMS Universal DL/I driver

The delete method removes existing segments from the database. Functionally the delete method in the IMS Universal DL/I driver is similar to the DL/I DLET call. The delete of a segment causes all of its child segments to be deleted too. The delete call must be preceded by a HOLD operation, which you can do using any one of either a getUnique, getNext, or getNextWithinParent method call. An IMS status code is returned by the delete method, which indicates the result of the DL/I operation. When a Path of segments is retrieved by the HOLD operation you can either delete all segments in that path or a subset of them. To delete them all, you call the delete method with no parameters. To delete segments from a point on the Path, other than the top to the bottom, use the delete method that takes an SSAList

argument and passes in an unqualified SSAList for the segment where you want deletion to begin. If you use a qualified SSAList, an exception is thrown.

Example 8-9 demonstrates how to delete all segments in a PATH. It shows how to use delete to selectively remove all MODEL segments and its dependent segments STOCK, STOCSALE, ORDER, and SALES, where the:

- DLRNO is 222222
- MAKE is FORD
- MODEL is FOCUS

Example 8-9 Deleting all segments in the path

```
SSAList ssaList = pcb.getSSAList("DEALER","STOCK");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"1234");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.addCommandCode(“MODEL",SSAList.CC_D);
Path path = ssaList.getPathForRetrieveReplace();
if (pcb.getUnique(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
       pcb.delete();
    }
}
while (pcb.getNext(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
       pcb.delete();
    }
}
```

Example 8-10 shows how to delete with an unqualified SSAList. The delete removes all STOCK segments and its logical child segment STOCSALE, where the:

- DLRNO is 222222
- MAKE is FORD
- MODEL is FOCUS

Example 8-10 Deleting segments with an unqualified ssalist

```
SSAList ssaList = pcb.getSSAList("DEALER","STOCK");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"1234");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.markAllFieldsForRetrieval("MODEL", true);
Path path = ssaList.getPathForRetrieveReplace();
SSAList stockSSAList = pcb.getSSAList("STOCK");
if (pcb.getUnique(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
       pcb.delete(stockSSAList);
    }
}
while (pcb.getNext(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
       pcb.delete(stockSSAList);
    }
}
```

Example 8-11 on page 191 is a total application showing all of the topics previously talked about in this chapter, which you can download from the file IMS Universal DL/I driver sample.zip, as described in Appendix D, “Additional material” on page 251.

Example 8-11 dlitest1 - A Complete DL/I application

```
package dlitest1;
import com.ibm.ims.dli.*;
public class DLIprogram {
   public static void main(String[] args) {
      PSB psb = null;
      PCB pcb = null;
      SSAList ssaList = null;
      Path path = null;
      PathSet pathSet = null;
      try {
         // establish a database connection
         IMSConnectionSpec connSpec
         = IMSConnectionSpecFactory.createIMSConnectionSpec();
         connSpec.setDatastoreName("IMSZ");
         connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
         connSpec.setPortNumber(5555);
         connSpec.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
         connSpec.setUser("userid");
         connSpec.setPassword("password");
         connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);

         psb = PSBFactory.createPSB(connSpec);
         System.out.println("**** Created a connection to the IMS database");
         pcb = psb.getPCB("AUTOLPCB");
         System.out.println("**** Created PCB object");
         // specify the segment search arguments
         ssaList = pcb.getSSAList("DEALER", "STOCK");
         // add the initial qualification
         // specify the fields to retrieve
         ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
         ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
         ssaList.markFieldForRetrieval("DEALER", "CITY", true);
         ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
         ssaList.markAllFieldsForRetrieval("MODEL", true);
         ssaList.markAllFieldsForRetrieval("STOCK", true);
         ssaList.markFieldForRetrieval("STOCK", "LOT", false);
         System.out.println("**** Created SSAList object");
         // obtain a Path containing the segments
         // that match the SSAList criteria
         path = ssaList.getPathForRetrieveReplace();
         System.out.println("**** Created Path object");
         // issue a DL/I GU call to retrieve the first segment on the Path
         if (pcb.getUnique(path, ssaList, true)) {
            System.out.println("Dealer Number: "+ path.getString("DEALER", "DLRNO"));
            System.out.println("Dealer Name:   "+ path.getString("DEALER", "DLRNAME"));
            System.out.println("City:          "+ path.getString("DEALER", "CITY"));
            System.out.println("ZIP:           "+ path.getString("DEALER", "ZIP"));
            System.out.println("Type of Model: "+ path.getString("MODEL", "MODTYPE"));
            System.out.println("Manufacturer:  "+ path.getString("MODEL", "MAKE"));
            System.out.println("Model:         "+ path.getString("MODEL", "MODEL"));
            System.out.println("Year:          "+ path.getString("MODEL", "YEAR"));
            System.out.println("MSRP:          "+ path.getString("MODEL", "MSRP"));
            System.out.println("# in Stock:    "+ path.getString("MODEL", "COUNT1"));
            System.out.println("VIN Number:    "+ path.getString("STOCK", "STKVIN"));
            System.out.println("Colour:        "+ path.getString("STOCK", "COLOR"));
            System.out.println("Price:         "+ path.getString("STOCK", "PRICE"));
            System.out.println("Warrenty:      "+ path.getString("STOCK", "WRNTY"));
         }
         // issue multiple DL/I GN calls until
```

```
// there are no more segments to retrieve
while (pcb.getNext(path, ssaList, true)) {
   System.out.println("Dealer Number: "+ path.getString("DEALER", "DLRNO"));
   System.out.println("Dealer Name:   "+ path.getString("DEALER", "DLRNAME"));
   System.out.println("City:          "+ path.getString("DEALER", "CITY"));
   System.out.println("ZIP:           "+ path.getString("DEALER", "ZIP"));
   System.out.println("Type of Model: "+ path.getString("MODEL", "MODTYPE"));
   System.out.println("Manufacturer:  "+ path.getString("MODEL", "MAKE"));
   System.out.println("Model:         "+ path.getString("MODEL", "MODEL"));
   System.out.println("Year:          "+ path.getString("MODEL", "YEAR"));
   System.out.println("MSRP:          "+ path.getString("MODEL", "MSRP"));
   System.out.println("# in Stock:    "+ path.getString("MODEL", "COUNT1"));
   System.out.println("VIN Number:    "+ path.getString("STOCK", "STKVIN"));
   System.out.println("Colour:        "+ path.getString("STOCK", "COLOR"));
   System.out.println("Price:         "+ path.getString("STOCK", "PRICE"));
   System.out.println("Warrenty:      "+ path.getString("STOCK", "WRNTY"));
}
// Insert a new DEALER, MODEL & 3 STOCK segments
ssaList = pcb.getSSAList("DEALER", "STOCK");
path = ssaList.getPathForInsert("DEALER");
path.setString("DLRNO", "7575");
path.setString("DLRNAME", "IBM Super Systems");
path.setString("CITY", "Portsmouth");
path.setString("ZIP", "PO6 3AU");
path.setString("PHONE", "0239256");
path.setString("MODTYPE", "MF");
path.setString("MODKEY", "IBM       Z10 GT    2010");
path.setString("MSRP", "12500");
path.setString("COUNT1", "3");
path.setString("STKVIN", "VT11234677098557729C");
path.setString("COLOR", "Blue");
path.setString("PRICE", "99999");
path.setString("LOT", "BS12345678");
path.setString("WRNTY", "Y");
short status = pcb.insert(path, ssaList);
if(status == IMSStatusCodes.BLANKS){
   System.out.println(" Insert of STOCK Segment 1 Successful");
} else {
   System.out.println(" Insert of STOCK Segment 1 FAILED");
}
ssaList = pcb.getSSAList("STOCK");
path = ssaList.getPathForInsert("STOCK");
path.setString("STKVIN", "VT11234677098557730C");
path.setString("COLOR", "Deep Blue");
path.setString("PRICE", "99999");
path.setString("LOT", "BS12345679");
path.setString("WRNTY", "Y");
status = pcb.insert(path, ssaList);
if(status == IMSStatusCodes.BLANKS){
   System.out.println(" Insert of STOCK Segment 2 Successful");
} else {
   System.out.println(" Insert of STOCK Segment 2 FAILED");
}
path.setString("STKVIN", "VT11234677098557731C");
path.setString("COLOR", "Light Blue");
path.setString("PRICE", "99999");
path.setString("LOT", "BS12345680");
path.setString("WRNTY", "Y");
int i = pcb.insert(path, ssaList);
if(i > 0){
```

```
   System.out.println(" Insert of STOCK Segment 3 Successful");
} else {
   System.out.println(" Insert of STOCK Segment 3 FAILED");
}
psb.commit();
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
      SSAList.EQUALS, "7575");
ssaList.addCommandCode("DEALER",SSAList.CC_D);
ssaList.addCommandCode("MODEL",SSAList.CC_D);
ssaList.addCommandCode("MODEL",SSAList.CC_P);
// specify the fields to retrieve
ssaList.markAllFieldsForRetrieval("DEALER", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("**** Created SSAList object");
// obtain a Path containing the segments
// that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("**** Created Path object");
// issue a DL/I GU call to retrieve the
// first segment on the Path
if (pcb.getUnique(path, ssaList, true)) {
   System.out.println("DEALER NUMBER:   "+ path.getString("DEALER", "DLRNO"));
   System.out.println("DEALER NAME:     "+ path.getString("DEALER", "DLRNAME"));
   System.out.println("CITY:            "+ path.getString("DEALER", "CITY"));
   System.out.println("ZIP:             "+ path.getString("DEALER", "ZIP"));
   System.out.println("PHONE NUMBER:    "+ path.getString("DEALER", "PHONE"));
   System.out.println("TYPE of MODEL:   "+ path.getString("MODEL", "MODTYPE"));
   System.out.println("MAKE MODEL YEAR: "+ path.getString("MODEL", "MODKEY"));
   System.out.println("MSRP:            "+ path.getString("MODEL", "MSRP"));
   System.out.println("Number in Stock: "+ path.getString("MODEL", "COUNT1"));
   System.out.println("VIN NUMBER:      "+ path.getString("STOCK", "STKVIN"));
   System.out.println("COLOR:           "+ path.getString("STOCK", "COLOR"));
   System.out.println("PRICE:           "+ path.getString("STOCK", "PRICE"));
   System.out.println("LOT:             "+ path.getString("STOCK", "LOT"));
   System.out.println("WARRENTY?:       "+ path.getString("STOCK", "WRNTY"));
}
while (pcb.getNextWithinParent(path, ssaList, true)) {
   System.out.println("VIN NUMBER:      "+ path.getString("STOCK", "STKVIN"));
   System.out.println("COLOR:           "+ path.getString("STOCK", "COLOR"));
   System.out.println("PRICE:           "+ path.getString("STOCK", "PRICE"));
   System.out.println("LOT:             "+ path.getString("STOCK", "LOT"));
   System.out.println("WARRENTY?:       "+ path.getString("STOCK", "WRNTY"));
}
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",SSAList.EQUALS, "7575");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markAllFieldsForRetrieval("STOCK", true);
ssaList.addCommandCode("MODEL",SSAList.CC_P);
// obtain a Path containing the segments
// that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("**** Get DLRNO 7575 ready for DELETE ****");
String blue = "Blue";
// issue a DL/I GU call to retrieve
```

```
   if (pcb.getUnique(path, ssaList, true)) {
   String color = path.getString("STOCK","COLOR").trim();
   System.out.println("Color is :'"+color+"'");
   if (color.equals(blue)) {
      System.out.println("STKVIN = "+ path.getString("STOCK", "STKVIN"));
      System.out.println("This one is Blue do not delete");}
   else{
      SSAList stockSSAList = pcb.getSSAList("STOCK");
      status = pcb.delete(stockSSAList);
      if(status == IMSStatusCodes.BLANKS){
         System.out.println(" DELETE of STOCK Segment "
      + path.getString("STOCK", "STKVIN")+" Successful");
      } else {
         System.out.println(" DELETE of STOCK Segment "
      + path.getString("STOCK", "STKVIN")+" FAILED");
      }
   }
}
while (pcb.getNextWithinParent(path, ssaList, true)) {
   String color = path.getString("STOCK","COLOR").trim();
   System.out.println("Color is :'"+color+"'");
   if (color.equals(blue)){
      System.out.println("This one is Blue so do not delete");
   } else {
      SSAList stockSSAList = pcb.getSSAList("STOCK");
      status = pcb.delete(stockSSAList);
      if(status == IMSStatusCodes.BLANKS) {
         System.out.println(" DELETE of STOCK Segment "
      + path.getString("STOCK", "STKVIN")+" Successful");
      } else {
         System.out.println(" DELETE of STOCK Segment "
      + path.getString("STOCK", "STKVIN")+" FAILED");
      }
   }
}
System.out.println(" All STOCK segments not COLOR = BLUE deleted");
ssaList = pcb.getSSAList("DEALER");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
      SSAList.EQUALS, "7575");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
path = ssaList.getPathForRetrieveReplace();
if (pcb.getUnique(path, ssaList, true)==true) {
   status = pcb.delete();
}
if(status == IMSStatusCodes.BLANKS) {
   System.out.println(" DEALER DLRNO = '7575' and dependents deleted");
} else {
   System.out.println(" DELETE PROBLEM");
}
psb.commit();

// specify the segment search arguments
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
      SSAList.EQUALS, "1234");
ssaList.addCommandCode("MODEL",SSAList.CC_D);
ssaList.addCommandCode("MODEL",SSAList.CC_P);
```

```
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markFieldForRetrieval("MODEL", "MAKE", false);
ssaList.markFieldForRetrieval("MODEL", "YEAR", false);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("**** Created SSAList object");
// obtain a Path containing the segments that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("**** Created Path object");
// issue a DL/I GU call to retrieve the segments on the Path
if (pcb.getUnique(path, ssaList, true)){
System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
System.out.println("MAKE, MODEL & YEAR: "+ path.getString("MODEL", "MODKEY"));
System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
}
// If data was retrieved update the values in fields MSRP in the MODEL
// segment and COLOR in the STOCK segment
if (pcb.getUnique(path, ssaList, true))
   path.setString("MSRP", "45999");
path.setString("COLOR", "Green");
pcb.replace(path);
if (pcb.getUnique(path, ssaList, true)){
System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
System.out.println("MAKE, MODEL & YEAR: "+ path.getString("MODEL", "MODKEY"));
System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
}
// Rollback the updates
psb.rollback();
// Get the data from the path for DLRNO 1234 to show that the
// updates have been rolled back.
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
      SSAList.EQUALS, "1234");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markFieldForRetrieval("MODEL", "MAKE", false);
ssaList.markFieldForRetrieval("MODEL", "YEAR", false);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("**** Created SSAList object");
// obtain a Path containing the segments that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
// issue a DL/I GU call to retrieve the segments on the Path
if (pcb.getUnique(path, ssaList, true)){
System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
```

```
         System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
         System.out.println("MAKE, MODEL & YEAR: "+ path.getString("MODEL", "MODKEY"));
         System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
         System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
         System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
         }
         // rollback the updates to MODEL & STOCK
         psb.rollback();
         // close the database connection
         psb.close();
         System.out.println("**** Disconnected from IMS database");
      } catch (DLIException e){
         System.out.println(e);
         System.exit(0);
      }
   }
}
```

Example 8-12 shows the output of the application in Example 8-11 on page 191.

Example 8-12 Output from the application

```
**** Created a connection to the IMS database
**** Created PCB object
**** Created SSAList object
**** Created Path object
Dealer Number: 1234
Dealer Name:   SAN JOSE FORD
City:          SAN JOSE
ZIP:           95777-3333
Type of Model: S
Manufacturer:  FORD
Model:         FOCUS
Year:          2002
MSRP:          17995
# in Stock:    03
VIN Number:    V234567890123456789V
Colour:        LIGHT BLUE
Price:         16000
Warrenty:      Y
 Insert of STOCK Segment 1 Successful
 Insert of STOCK Segment 2 Successful
 Insert of STOCK Segment 3 Successful
**** Created SSAList object
**** Created Path object
DEALER NUMBER:   7575
DEALER NAME:     IBM Super Systems
CITY:            Portsmouth
ZIP:             PO6 3AU
PHONE NUMBER:    0239256
TYPE of MODEL:   MF
MAKE MODEL YEAR: IBM       Z10 GT    2010
MSRP:            12500
Number in Stock: 3
VIN NUMBER:      VT11234677098557729C
COLOR:           Blue
PRICE:           99999
LOT:             BS12345678
WARRENTY?:       Y
VIN NUMBER:      VT11234677098557730C
```

```
COLOR:           Deep Blue
PRICE:           99999
LOT:             BS12345679
WARRENTY?:       Y
VIN NUMBER:      VT11234677098557731C
COLOR:           Light Blue
PRICE:           99999
LOT:             BS12345680
WARRENTY?:       Y
**** Get DLRNO 7575 ready for DELETE ****
Color is :'Blue'
STKVIN = VT11234677098557729C
This one is Blue do not delete
Color is :'Deep Blue'
 DELETE of STOCK Segment VT11234677098557730C Successful
Color is :'Light Blue'
 DELETE of STOCK Segment VT11234677098557731C Successful
 All STOCK segments not COLOR = BLUE deleted
 DEALER DLRNO = '7575' and dependents deleted
**** Created SSAList object
**** Created Path object
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:               SAN JOSE
MAKE, MODEL & YEAR: FORD      FOCUS     2002
MSRP:               17995
VIN NUMBER:         V234567890123456789V
COLOR:              LIGHT BLUE
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:               SAN JOSE
MAKE, MODEL & YEAR: FORD      FOCUS     2002
MSRP:               45999
VIN NUMBER:         V234567890123456789V
COLOR:              Green
**** Created SSAList object
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:               SAN JOSE
MAKE, MODEL & YEAR: FORD      FOCUS     2002
MSRP:               17995
VIN NUMBER:         V234567890123456789V
COLOR:              LIGHT BLUE
**** Disconnected from IMS database
```

8.1.6 Using the batch methods with the IMS Universal DL/I driver

When you hear that there is a batch method of using the DL/I driver, you might immediately think it is something that you ran in batch using System z Job Control Language (JCL). You will soon find out that it is wrong.

Batch method is a way of accessing or updating multiple segments with a single call. Instead of the application having to make multiple getUnique and getNext calls, IMS performs all of the calls and returns the results back to the client in a single batch operation. The number of rows to be returned for each batch network operation is determined by the fetch size property, which is set for you internally but can be overridden by the pcb.getFetchSize(n); statement, where n is a number you deem acceptable. This is especially relevant for a distributed client to maximize network efficiency. The driver builds a request for the number of rows asked for to

be returned and sends it to ODBM (using IMS Connect) who interacts with IMS to retrieve this number of rows (if available). Network interaction retrieves multiple rows (or segments). If the remote client application continues to ask for more rows, the driver submits a request for another set of rows to be returned. This facility is available in all of the drivers:

- Universal DB Resource Adapter – for JDBC, and for CCI SQL or DL/I access
- Universal JDBC Driver
- Universal DL/I Driver

Example 8-13 shows an application that uses the batch methods for retrieval, updating, and deleting data.You can download this example from the file IMS Universal DL/I driver sample.zip, as described in Appendix D, “Additional material” on page 251.

Example 8-13 dlitest2 - Batch methods example

```
package dlitest2;
import com.ibm.ims.dli.*;
public class DLIprogram2 {
   public static void main(String[] args) {
      PSB psb = null;
      PCB pcb = null;
      SSAList ssaList = null;
      Path path = null;
      PathSet pathSet = null;
      int cnt = 0;
      try {
         // establish a database connection
         IMSConnectionSpec connSpec
         = IMSConnectionSpecFactory.createIMSConnectionSpec();
         connSpec.setDatastoreName("IMSZ");
         connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
         connSpec.setPortNumber(5555);
         connSpec.setMetadataURL(
         "class://samples.ims.openDb.AUTPSB11DatabaseView");
         connSpec.setUser("IMS2R");
         connSpec.setPassword("tObyjugs");
         connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
         psb = PSBFactory.createPSB(connSpec);
         System.out.println(
               "**** Created a connection to the IMS database");
         pcb = psb.getPCB("AUTOLPCB");
         System.out.println("**** Created PCB object");
      // add the initial qualification
      // specify the segment search arguments
         ssaList = pcb.getSSAList("DEALER","MODEL");
         ssaList.addInitialQualification("DEALER","DLRNAME",
               SSAList.NOT_EQUAL,"IBM Bristol Cars              ");
         ssaList.addInitialQualification("MODEL","MODEL",
               SSAList.EQUALS,"FOCUS");
         ssaList.appendQualification("MODEL",SSAList.AND,
               "COUNT1",SSAList.GREATER_THAN, "02");
         // add the initial qualification
         // specify the fields to retrieve
         ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
         ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
         ssaList.markFieldForRetrieval("DEALER", "CITY", true);
         ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
```

```
            ssaList.markAllFieldsForRetrieval("MODEL", true);
//          ssaList.markAllFieldsForRetrieval(3, true);
            System.out.println("**** Created SSAList object");
            // obtain a Path containing the segments
            // that match the SSAList criteria
            System.out.println("**** Created Path object");
            // issue a DL/I GU call to retrieve the first segment on the Path
            pathSet = pcb.batchRetrieve(ssaList);
            while(pathSet.hasNext()){
               path = pathSet.next();
               cnt = cnt + 1;
               System.out.println("Dealer Number: "
                     + path.getString("DEALER", "DLRNO")
                     + "  Dealer Name:   "
                     + path.getString("DEALER", "DLRNAME")
                       + "  City:          "
                     + path.getString("DEALER", "CITY")
                       + "  ZIP:           "
                     + path.getString("DEALER", "ZIP"));
               System.out.println("Model Type: "
                     + path.getString("MODEL", "MODTYPE")
                       + "  Manufacturer: "
                     + path.getString("MODEL", "MAKE")
                       + "  Model: "
                     + path.getString("MODEL", "MODEL")
                       + "  Year: "
                     + path.getString("MODEL", "YEAR")
                       + "  MSRP: "
                     + path.getString("MODEL", "MSRP")
                       + "  # in Stock: "
                     + path.getString("MODEL", "COUNT1"));
            }
            System.out.println("Number of loops = "+ cnt);
            cnt = cnt - cnt;
            ssaList = pcb.getSSAList("DEALER","MODEL");
            ssaList.addInitialQualification("DEALER","DLRNAME",
                  SSAList.NOT_EQUAL,"IBM Bristol Cars              ");
            ssaList.addInitialQualification("MODEL","MODEL",
                  SSAList.EQUALS,"FOCUS");
            ssaList.appendQualification("MODEL",SSAList.AND,
                  "COUNT1",SSAList.GREATER_THAN, "02");
            path = ssaList.getPathForBatchUpdate("MODEL");
            path.setString("MSRP", "15500");
            int i = pcb.batchUpdate(path, ssaList);
            System.out.println("Number of Segments Updated = "+ i);
            System.out.println("Updates Done");
//          pathSet = pcb.batchRetrieve(ssaList);
            // issue multiple DL/I GN calls until
            // there are no more segments to retrieve
            ssaList = pcb.getSSAList("DEALER","MODEL");
            ssaList.addInitialQualification("DEALER","DLRNAME",
                  SSAList.NOT_EQUAL,"IBM Bristol Cars              ");
            ssaList.addInitialQualification("MODEL","MODEL",
                  SSAList.EQUALS,"FOCUS");
            ssaList.appendQualification("MODEL",SSAList.AND,
```

```
      "COUNT1",SSAList.GREATER_THAN, "02");
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
pathSet = pcb.batchRetrieve(ssaList);
while(pathSet.hasNext()){
   path = pathSet.next();
   cnt = cnt + 1;
   System.out.println("Dealer Number: "
         + path.getString("DEALER", "DLRNO")
           + "  Dealer Name:   "
         + path.getString("DEALER", "DLRNAME")
           + "  City:          "
         + path.getString("DEALER", "CITY")
           + "  ZIP:           "
         + path.getString("DEALER", "ZIP"));
   System.out.println("Model Type: "
         + path.getString("MODEL", "MODTYPE")
           + "  Manufacturer: "
         + path.getString("MODEL", "MAKE")
           + "  Model: "
         + path.getString("MODEL", "MODEL")
           + "  Year: "
         + path.getString("MODEL", "YEAR")
           + "  MSRP: "
         + path.getString("MODEL", "MSRP")
           + "  # in Stock: "
         + path.getString("MODEL", "COUNT1"));
}
System.out.println("Numer of loops = "+ cnt);
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.addInitialQualification("DEALER","DLRNAME",
      SSAList.NOT_EQUAL,"IBM Bristol Cars              ");
ssaList.addInitialQualification("MODEL","MODEL",
      SSAList.EQUALS,"FOCUS");
ssaList.appendQualification("MODEL",SSAList.AND,
      "COUNT1",SSAList.GREATER_THAN, "02");
i = pcb.batchDelete(ssaList);
System.out.println("Number of Segments Deleted = "+ i);
System.out.println("Deletes Done");
cnt = cnt - cnt;
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
pathSet = pcb.batchRetrieve(ssaList);
while(pathSet.hasNext()){
   path = pathSet.next();
   cnt = cnt + 1;
   System.out.println("Dealer Number: "
         + path.getString("DEALER", "DLRNO")
```

```
                  + "  Dealer Name:   "
                + path.getString("DEALER", "DLRNAME")
                  + "  City:          "
                + path.getString("DEALER", "CITY")
                  + "  ZIP:           "
                + path.getString("DEALER", "ZIP"));
            System.out.println("Model Type: "
                + path.getString("MODEL", "MODTYPE")
                  + "  Manufacturer: "
                + path.getString("MODEL", "MAKE")
                  + "  Model: "
                + path.getString("MODEL", "MODEL")
                  + "  Year: "
                + path.getString("MODEL", "YEAR")
                  + "  MSRP: "
                + path.getString("MODEL", "MSRP")
                  + "  # in Stock: "
                + path.getString("MODEL", "COUNT1"));
         }
         System.out.println("Numer of loops = "+ cnt);
         // rollback the updates to MODEL
         psb.rollback();
         // close the database connection
         psb.close();
         System.out.println("**** Disconnected from IMS database");
      } catch (DLIException e){
         AIB aib = e.getAib();
         if (aib != null) {
            String sc = aib.getDBPCB().getStatusCodeChars();
            String retcode = aib.getReturnCodeHex();
            String reascode = aib.getReasonCodeHex();
            System.out.println("Status code: " + sc + " Return Code: "
                  + retcode + " Reason Code: " + reascode);
         }
         System.out.println(e);
         System.exit(0);
      }
   }
}
```

8.2 Writing applications with the IMS Universal DB Resource Adapter and the CCI programming approach

Using the IMS Universal DB Resource Adapter, you can write applications using the CCI programming model. JDBC is our recommended approach for accessing IMS data with the IMS Universal Drivers, but there can be advantages using the CCI programming model. JDBC does not allow you the granularity of accessing hierarchical data that DL/I allows. The CCI programming model allows you to use SQL and DL/I syntax within one connection and application. You can also use the stand-alone JDBC and stand-alone DL/I drivers within one application, but you need two separate connections, and you must coordinate the commit or rollbacks yourself (or with global transaction support).

The IMS Universal DB Resource Adapters are intended to be used in managed JEE Application Servers, but they also give you the possibility of using them in stand-alone non-managed applications.

The IMS Universal DB Resource Adapters are RAR files and contain jar files. To use the IMS Universal DB Resource Adapters in stand-alone applications, it is useful to extract the jar files using an extract utility that can extract RAR archives. The extracted Jar files must be (as with all other drivers) in the class path of the application.

The example in this section provides the coding of an application using the IMS Universal DB Resource Adapter and the CCI programming approach in a stand-alone environment.

For more information about the CCI API, see the CCI API in the Information Center at:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.apr/ims_odbjcasupportforcci.htm

8.2.1 Writing the application step-by-step

For a full reference of the program see Appendix D, “Additional material” on page 251. To write the application:

1. You need the import statements that are listed in Example 8-14 in your application.

Example 8-14 Import Statements of CCI Application

```
import javax.naming.InitialContext; //only in managed environemnts
import javax.transaction.UserTransaction; //only in manged environments
import javax.resource.cci.*;
import com.ibm.ims.db.cci.*;
```

   Therefore you must have the WebSphere Application Library in your class path because it contains the j2ee.jar, which contains the javax.* classes used.

2. In a non-managed environment, you must create a new IMSManagedConnectionFactory object and specify the needed values for the connection. From the ManagedConnectionFactory, Example 8-15, you can create a ConnectionFactory that creates the connection itself.

Example 8-15 Create ManagedConnectionFactory

```
IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
mcf.setUser("user");
mcf.setPassword("password");
mcf.setDatastoreName("IMS2");
mcf.setDatastoreServer("myhost.itso.ibm.com");
mcf.setPortNumber(5555);
mcf.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
mcf.setSSLConnection(false);
mcf.setDriverType(IMSManagedConnectionFactory.DRIVER_TYPE_4);
mcf.setLoginTimeout(10);
ConnectionFactory cf =  (ConnectionFactory) mcf.createConnectionFactory();
Connection conn = cf.getConnection();
```

   In a managed environment, you just lookup the JNDI name of the ConnectionFactory with the help of the InitialContext, which is passed to the application from the container, as shown in Example 8-16 on page 203.

Example 8-16 Lookup MCF

```
InitialContext ic = new InitialContext();
ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
Connection conn = cf.getConnection();
```

3. For transaction handling, you can use the code in Example 8-17 in your non-managed application.

Example 8-17 Transaction calls

```
LocalTransaction trans =conn.getLocalTransaction();
trans.begin();
...
trans.commit(); // or trans.rollback();
```

   In a managed environment, you can get the UserTransaction object from the SessionContext, as shown in 5.3.1, “JCA/Common Client Interface approach” on page 110.

4. Create an Interaction object and a SQLInteractionspec and a DLIInteractionSpec object that contains information about the Interaction itself, as shown in Example 8-18.

Example 8-18 Create Interaction and InteractionSpecs

```
Interaction ix = conn.createInteraction();
SQLInteractionSpec sqlSpec = new SQLInteractionSpec();
DLIInteractionSpec dliSpec = new DLIInteractionSpec();
```

5. Insert one DEALER root segment and three dependent MODEL segments with the SQL syntax shown in Example 8-19.

Example 8-19 Insert segments with SQL

```
sqlSpec.setSQL("INSERT INTO AUTOLPCB.DEALER (DLRNO, ZIP, DLRNAME, CITY, PHONE)  " +
   "VALUES('7777', '70565','Thilos A,B and X Model Cars','Stuttgart','555-888')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
   " VALUES ('7777','S','LIDLA','A Normal','2010','20000','05')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
   "VALUES ('7777', 'M','LIDLA','B Plus','2010','40000','03')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
   "VALUES ('7777','L','LIDLA','X Large','2010','60000','01')");
ix.execute(sqlSpec, null);
```

6. Retrieve the information from the database. With the DL/I syntax, the code looks like Example 8-20.

Example 8-20 Retrieve information with DL/I

```
dliSpec.setFunctionName("RETRIEVE");
dliSpec.setPCBName("AUTOLPCB");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL ");
ResultSet dlirs = (ResultSet) ix.execute(dliSpec, null);
while (dlirs.next()) {
   System.out.print(dlirs.getString("DLRNO")+" ; ");
```

```
   System.out.print(dlirs.getString("DLRNAME")+" ; ");
   System.out.print(dlirs.getString("MODTYPE")+" ; ");
   System.out.print(dlirs.getString("MAKE")+" ; ");
   System.out.print(dlirs.getString("MODEL")+" ; ");
   System.out.print(dlirs.getString("MSRP")+" ; ");
   System.out.print(dlirs.getString("COUNT1")+" \n");
}
```

In our example, the RETRIEVE function is used by specifying an IMS Path Call in the SSA statement. The execute function does not need an input object. As an output object you will get back a ResultSet object.

The same function with an SQL syntax looks like Example 8-21.

Example 8-21 Retrieve information with SQL

```
sqlSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER,AUTOLPCB.MODEL " +
      "WHERE DEALER.DLRNO='7777' AND MODEL.DEALER_DLRNO='7777'");
ResultSet sqlrs = (ResultSet) ix.execute(sqlSpec, null);
while (sqlrs.next()) {
   System.out.print(sqlrs.getString("DLRNO")+" ; ");
   System.out.print(sqlrs.getString("DLRNAME")+" ; ");
   System.out.print(sqlrs.getString("MODTYPE")+" ; ");
   System.out.print(sqlrs.getString("MAKE")+" ; ");
   System.out.print(sqlrs.getString("MODEL")+" ; ");
   System.out.print(sqlrs.getString("MSRP")+" ; ");
   System.out.print(sqlrs.getString("COUNT1")+" \n");
}
```

It uses the WHERE clause to specify which segments are to be received and gets a ResultSet that can be used in the usual way.

7. One thing that cannot be done with SQL syntax is for example to update the last dependent segment in the database. For this call we use the DL/I syntax in Example 8-22.

Example 8-22 Update information with DL/I

```
dliSpec.setFunctionName("UPDATE");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL *L");
RecordFactory rf = cf.getRecordFactory();
MappedRecord input = rf.createMappedRecord("DEALER");
input.put("DEALER.DLRNAME", "Thilos A and B Model Cars");
input.put("MODEL.MSRP", "OOOOO");
input.put("MODEL.COUNT1", "OO");
ix.execute(dliSpec, input);
```

The SSA contains the *L command and uses a path call from DEALER and MODEL by specifying the *D command. In this case, you must create a MappedRecord with DEALER as parameter, but as it is a path call, you can modify any segment in the path that is specified. This example sets the last occurrence of MODEL in this path to MSRP=00000 and COUNT1=00 to indicate that this model is no longer available. Therefore we change also the DLRNAME in the parent DEALER segment. If you just want to update the MODEL segment, you do not have to specify a *D path call command and can map the record to the MODEL segment.

Note: The last occurrence is not necessarily the last inserted. It depends on the key field of the segment.

The key field of the Model segment is built from the MAKE, MODEL, and YEAR. So the MODKEY fields looks like Example 8-23 for our three inserts.

Example 8-23 Output of MODKEY retrieve

```
MODKEY No 1 is “LIDLA     A Normal  2010”
MODKEY No 2 is “LIDLA     B Plus    2010”
MODKEY No 3 is “LIDLA     X Large   2010”
```

The last occurrence is the last one because the key differentiates the position on A, B, or X.

8. In the end, we do a delete of the inserted DEALER segment with the code in Example 8-24.

Example 8-24 Delete data with SQL

```
sqlSpec.setSQL("DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='7777'");
sqlrs = (ResultSet) ix.execute(sqlSpec, null);
if(sqlrs.next()){
   System.out.println("DEALER Rows deleted :"+sqlrs.getInt("UPDATE_COUNT"));
}
```

With the deletion of the DEALER segment, also all referenced (dependent) segments are deleted. To get the information, if the delete was successful, you can use the ResultSet and get the generated UPDATE_COUNT field.

8.2.2 Complete code example of the CCI mixed application

The code in Example 8-25 shows the complete application, which we described in 8.2.1, “Writing the application step-by-step” on page 202.

Example 8-25 CCIStandaloneDLIandSQL

```
package samples.ims.applications;
import javax.resource.cci.*;
import com.ibm.ims.db.cci.*;

public class CCIStandaloneDLIandSQL {
   public static void main(String[] args) {
      System.out.println("Creating Connection Factory and specifing values");
      Connection conn = null;
      IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
      mcf.setUser("user");
      mcf.setPassword("password");
      mcf.setDatastoreName("IMS2");
      mcf.setDatastoreServer("myhost.itso.ibm.com");
      mcf.setPortNumber(5555);
      mcf.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
      mcf.setSSLConnection(false);
      mcf.setDriverType(IMSManagedConnectionFactory.DRIVER_TYPE_4);
      mcf.setLoginTimeout(10);
      try{
      ConnectionFactory cf =  (ConnectionFactory) mcf.createConnectionFactory();
      conn = cf.getConnection();
      LocalTransaction trans =conn.getLocalTransaction();
      trans.begin();
      Interaction ix = conn.createInteraction();
      System.out.println("Started Transaction");
      SQLInteractionSpec sqlSpec = new SQLInteractionSpec();
```

```
System.out.println("Insert DEALER segment and 3 MODELs");
sqlSpec.setSQL("INSERT INTO AUTOLPCB.DEALER " +
   "(DLRNO, ZIP, DLRNAME, CITY, PHONE)  " +
   "VALUES('7777', '70565','Thilos A,B and X Model Cars','Stuttgart','555-888')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
   " VALUES ('7777','S','LIDLA','A Normal','2010','20000','05')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
   "VALUES ('7777', 'M','LIDLA','B Plus','2010','40000','03')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
   "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
   "VALUES ('7777','L','LIDLA','X Large','2010','60000','01')");
ix.execute(sqlSpec, null);
System.out.println("Display DEALER and MODEL with DL/I");
DLIInteractionSpec dliSpec = new DLIInteractionSpec();
dliSpec.setFunctionName("RETRIEVE");
dliSpec.setPCBName("AUTOLPCB");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL ");
ResultSet dlirs = (ResultSet) ix.execute(dliSpec, null);
while (dlirs.next()) {
   System.out.print(dlirs.getString("DLRNO")+" ; ");
   System.out.print(dlirs.getString("DLRNAME")+" ; ");
   System.out.print(dlirs.getString("MODTYPE")+" ; ");
   System.out.print(dlirs.getString("MAKE")+" ; ");
   System.out.print(dlirs.getString("MODEL")+" ; ");
   System.out.print(dlirs.getString("MSRP")+" ; ");
   System.out.print(dlirs.getString("COUNT1")+" \n");
}
System.out.println("Modify Last Occurence of MODEL of the inserted DEALER");
dliSpec.setFunctionName("UPDATE");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL *L");
RecordFactory rf = cf.getRecordFactory();
MappedRecord input = rf.createMappedRecord("DEALER");
input.put("DEALER.DLRNAME", "Thilos A and B Model Cars");
input.put("MODEL.MSRP", "00000");
input.put("MODEL.COUNT1", "00");
ix.execute(dliSpec, input);
sqlSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER,AUTOLPCB.MODEL " +
   "WHERE DEALER.DLRNO='7777' AND MODEL.DEALER_DLRNO='7777'");
ResultSet sqlrs = (ResultSet) ix.execute(sqlSpec, null);
while (sqlrs.next()) {
   System.out.print(sqlrs.getString("DLRNO")+" ; ");
   System.out.print(sqlrs.getString("DLRNAME")+" ; ");
   System.out.print(sqlrs.getString("MODTYPE")+" ; ");
   System.out.print(sqlrs.getString("MAKE")+" ; ");
   System.out.print(sqlrs.getString("MODEL")+" ; ");
   System.out.print(sqlrs.getString("MSRP")+" ; ");
   System.out.print(sqlrs.getString("COUNT1")+" \n");
}
sqlSpec.setSQL("DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='7777'");
sqlrs = (ResultSet) ix.execute(sqlSpec, null);
if(sqlrs.next()){
   System.out.println("DEALER Rows deleted :"+sqlrs.getInt("UPDATE_COUNT"));
}
trans.rollback();
dlirs.close();
```

```
      sqlrs.close();
      ix.close();
      conn.close();
      System.out.println("Transaction rolled back and Connection closed");
      } catch (Exception e) {
         e.printStackTrace();
         try {
            conn.close();
         } catch (Exception e1) {
            e1.printStackTrace();
         }
      }
   }
}
```

The output of the application looks like the lines in Example 8-26.

Example 8-26 Output of CCIStandaloneSQLandDLI Java application

```
Creating Connection Factory and specifing values
Started Transaction
Insert DEALER segment and 3 MODELs
Display DEALER and MODEL with DL/I
7777 ; Thilos A,B and X Model Cars    ; S  ; LIDLA      ; A Normal   ; 20000 ; 05
7777 ; Thilos A,B and X Model Cars    ; M  ; LIDLA      ; B Plus     ; 40000 ; 03
7777 ; Thilos A,B and X Model Cars    ; L  ; LIDLA      ; X Large    ; 60000 ; 01
Modify Last Occurence of MODEL of the inserted DEALER
7777 ; Thilos A and B Model Cars      ; S  ; LIDLA      ; A Normal   ; 20000 ; 05
7777 ; Thilos A and B Model Cars      ; M  ; LIDLA      ; B Plus     ; 40000 ; 03
7777 ; Thilos A and B Model Cars      ; L  ; LIDLA      ; X Large    ; 00000 ; 00
DEALER Rows deleted :1
Transaction rolled back and Connection closed
```

9

Operational considerations

This chapter covers general recommendations and best practices. It contains information about the following topics:

- Architectural suggestions
- Enhancing existing applications
- Tracing in problem cases
- Using tools with IMS Open Database
- Additional sample programs

9.1 Architectural suggestions

We put together some information that can help you to find the right solution for your requirement.

9.1.1 Application middle layer

Sometimes it can make sense to develop a middle layer to prevent application developers from accessing IMS databases directly. Such cases are:

- Usually the application developers for distributed environments do not have a valid user ID and password on the z/OS side to access IMS resources directly. Handling the security for all developers involved in a large application development project can be a large effort.
- A technical user ID can help, but in pre-production environments, some databases or parts of it must only be accessible to a certain group. Sometimes this security is currently implemented by the IMS transaction, but with this solution you bypass the transaction. A middle layer that implements this security again can help in this case. The alternative is to specify different PSBs for the database with different access levels.
- An application middle layer can also help to prevent heavy database workloads in IMS. Sometimes unqualified database calls can cause a database scan and causes much I/O work on the database, which can affect performance for other transactions and users. A middle layer can help to allow only specific database calls to IMS.

Such a middle layer can be implemented in various ways:

- One option is to create Web Services for application developers to do specific calls against an IMS database. This approach makes it possible to switch easily to different databases and allows a looser coupling for a service-oriented approach.
- Another option is to implement a persistence layer, such as Hibernate or JPA, for your application. The application developer uses, in this case, only Java objects in the application, and the database calls are done by the persistence layer under the covers. Because the IMS Open Database feature uses industry standards, this approach is easy to implement with small changes to the current restriction or supported SQL syntax of the IMS Universal Drivers.
- Several more options exist, such as proxing the request through an EJB bean or a self written solution.

Note: An alternative to implementing the middle layer is to use the CSL ODBM user exits to allow or disallow certain incoming statements or manipulation of the outcome of the statements. See “ODBM user exits” on page 55 for more information.

A general recommendation cannot be given because the approach refers to the specific requirements of the application and company. With the Open Database feature, IMS 11 is open to the complete distributed world with all chances, and the security considerations must be resolved before using it in production.

9.1.2 Sysplex considerations

We did not focus on a sysplex environment in this book to keep the scenarios and the explanation simple. There are several things you must keep in mind for the IMSPlex itself. If you have Data Sharing in the IMSPlex in place, it does not matter to which IMS the Open DB

request is routed. You can duplicate each component for backup or workload balancing reasons.

Figure 9-1 shows an example system environment with two IMS systems in a IMSPlex with data sharing enabled. Each LPAR has its own IMS Connect Address space (IMSHWSA and IMSHWSB) and ODBM address space (IMSODA and IMSODB). IMS Connect communicates with ODBM within the LPAR using a local PC call, but IMS Connect can also communicate across LPAR boundaries through the Structured Call Interface (SCI), which supports XCF communication. So you just need one IMS Connect address space (IMSHWSB), which routes the request depending on the ODBM configuration to IMSODB or IMSODA. To avoid a single point-of-failure in the IMS Connect address space, it is possible to use the z/OS Sysplex Distributor to route the requests to a different IMS Connect address space (here IMSHWSA) in case of a failure.

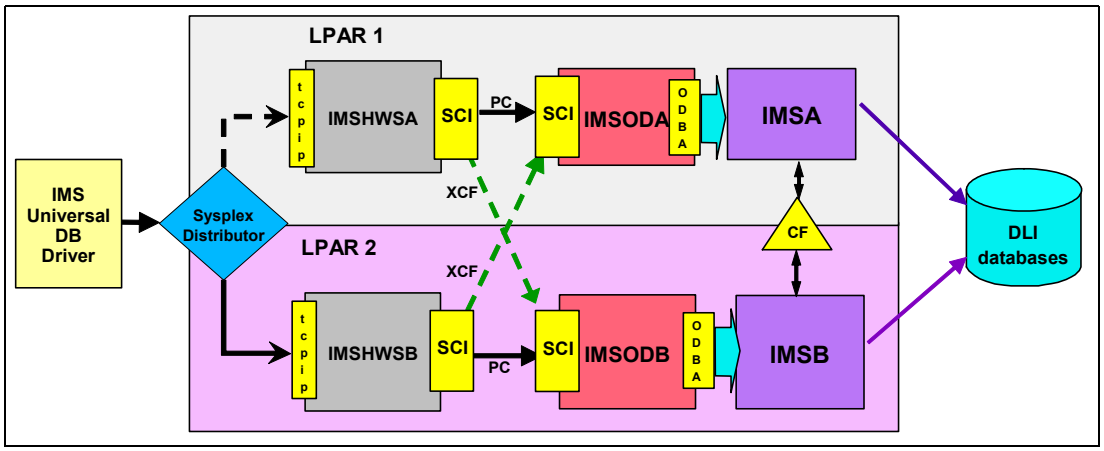


Figure 9-1 A two member IMSPlex sample environment

You must have one ODBM task per LPAR, which has a control region. One ODBM can connect to more than one control region in the same IMSplex on the same LPAR.

The IMS PROCLIB member CSLDCxxx can be shared between the ODBMs in a Sysplex. Eeach ODBM can also have its own configuration member. The configuration member has two parts:

- A global section, which sets values for the IMSplex as a whole.
- A local section, which defines the various ODBM address spaces and control regions, which overrides the global section.

The two IMS Connects and each IMS must belong to the same IMSPLEX (CSL group). When IMS Connect starts up, a connection is made to all members of the CSL group, and information is exchanged so that IMS Connect acquires knowledge of the real ODBM address spaces and the database component of the DBCTL and DB/DC systems they are in connection with; therefore, they can be addressed for IMS Universal Driver queries.

For a detailed example with configuration members for an IMS Open Database Sysplex environment, see *IMS Version 11 Technical Overview,* SG24-7807.

9.1.3 Performance considerations

Performance is a tough topic to deal with. The focus in this book is not on performance aspects. The performance is strongly influenced by the design of your queries in combination with the database layout. Configuration parameters and application parameters are also

important and can influence performance. We want to highlight the parameters that can influence performance:

- IMS Universal Driver-related performance:
  - When possible, use type-2 connectivity instead of type-4 because you can save the communication across TCP/IP and IMS Connect. When you use WebSphere Application Server for z/OS on the same LPAR as IMS, you can also use the RRSLocalOption of the IMS Database Resource Adapter.
  - Use Local transaction support instead of using XA transaction support where possible because you will save the expensive two-phase commit processing.
  - Return only needed columns in SELECT statements by specifying them instead of using the star (*) value.
  - Compare columns to specific values instead of other columns.
  - Use PreparedStatements if you are doing several queries or updates because the IMS Universal Drivers do not have to build the SSA list for the DL/I call each time and just have to insert the values.
  - Increase the fetchSize() parameter to get more results back with each network call, which can increase performance for large queries because you save several network calls.
  - Use secondary indexes and maybe reorganize your database where possible to fit the requirements of the application, especially for batch using applications.
  - Use DL/I for specific requests that JDBC and SQL cannot do, such as a get next parent call.
- IMS Connect and ODBM-related performance:
  - Increase the MAXTHREAD parameter in the CSLDCxxx member to allow more parallel access to your database.
  - Distribute your requests to several IMSplex members where possible.
  - Use PC calls instead of XCF calls, which means that when possible, have IMS Connect or the type-2 application and ODBM with IMS on the same LPAR.
- RRS logging performance:
  - Because of the use of RRS, ODBA performance is related to the RRS logging performance. RRS uses z/OS logger and five log streams that can be shared by multiple systems in a sysplex. You have several choices of the z/OS logger implementation. Consider that your choice must meet your performance and recovery requirements. For a description of configuring and defining RRS logging requirements, see *z/OS V1R6.0 MVS™ Programming: Resource Recovery*, SA22-7616.

For database performance, use the traditional procedures that you have for your daily IMS database maintenance to keep your IMS database healthy, such as regular reorganizations. There are no special requirements necessary for the IMS Open Database feature.

9.2 Enhancing existing applications

Using the IMS Open Database feature and the IMS Universal Database Drivers, you can enhance your existing environments.

9.2.1 ODBA access through ODBM

The former architecture of the IMS DB Resource Adapter needed WebSphere Application Server for z/OS to be on the same LPAR as IMS. This solution uses ODBA to get to the DL/I data. You do not require a DRA module to access IMS Open Database through IMS Connect. But you do not need to redevelop your application because the IMS Universal DB Resource Adapter is downward compatible. The ODBA interface from previous versions of IMS can coexist with IMS 11 without modification, but we recommend using the new features.

ODBA can make cross address space calls (PC calls) but only within the same logical partition. The ODBA modules are loaded into the address space of the application, which is running under its container's TCB, which poses a potential problem because if WebSphere Application Server (with ODBA within it) is terminated while an application's DL/I call is in process, the IMS control region receives an abend U113 when it detects that the ODBA TCB is no longer around.

Add the IMSPLEX= parameter to your existing DRAs and reassemble them. This information is what provides u113 isolation to ODBA applications because it causes ODBA to use ODBM instead of connecting directly to IMS. You might also want to add the ODBMNAME= parameter if you want to ensure connection to a specific ODBM. Example 9-1 shows a sample DRA.

Example 9-1 Sample DRA for ODBA access through ODBM

```
//DFSIVP10 EXEC PROC=ASMDRA,MBR=DFSIMSB0
//ASM.SYSIN DD *
DFSIMSB0 CSECT
         DFSPRP DSECT=NO,                                              X
            FUNCLV=2,                 ODBA FUNCTION LEVEL              X
            DDNAME=DFSDB2SP,          DDNAME FOR DRA RESLIB            X
            DSNAME=IMS11B.SDFSRESL,   DSNAME FOR DRA RESLIB            X
            DBCTLID=IMSB,             DBCTL IDENTIFIER                 X
            USERID=,                  USER IDENTIFIER                  X
            MINTHRD=1,                MINIMUM NUMBER OF THREADS        X
            MAXTHRD=1,                MAXIMUM NUMBER OF THREADS        X
            TIMER=60,                 IDENTIFY TIMER VALUE DEFAULT     X
            FPBUF=,                   NUMBER OF FP BUFFERS PER THREAD  X
            FPBOF=,                   NUMBER OF FP OVERFLOW BUFFERS    X
            SOD=A,                    SNAP DATASET OUTPUT CLASS        X
            TIMEOUT=60,               DRATERM TIMEOUT VALUE            X
            IDRETRY=0,                IDENTIFY RETRY COUNT             X
            CNBA=,                    TOTAL FP NBA BUFFERS FOR CCTL    X
            IMSPLEX=PLEXB,            IMSPLEX NAME                     X
            ODBMNAME=ODOB             OPTIONAL ODBMNAME
END
//*
```

Although the ODBA program connects to IMS through ODBM, connecting to an ODBM is only possible if it is running on the same LPAR. If your programs must access an ODBM on another LPAR, you can choose between implementing your own DRDA solution or using the ODBM API.

For more information about using the ODBA interface see section *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity,* SG24-6794.

9.2.2 Enabling unsupported Java environments

The IMS Universal JDBC Drivers work in most Java-using applications that support the standards, such as JDBC. There are certainly some restrictions because the drivers currently only support what is possible with IMS. It is not possible to create a database or table using an SQL statement, for example.

IMS has no catalog where the metadata information is stored, which is needed by the IMS Universal drivers. Therefore you have to use the metadata class file that is normally packaged in a Jar file. Some applications support JDBC, but you can select just one Jar file for the driver. To get the driver operating in such environments, you have different options:

- You can pack your generated metadata files in the driver jar file, and then you can reference it in the application. The disadvantage of this solution is that you must edit the driver each time there is a new driver version to deploy or you change or add any database.
- Another approach is to add the metadata file to the class path of the application. Therefore you have two options, depending on the architecture of the application you want to enable:
  - You can specify the class path of the application to contain the Jar files of the IMS Universal Driver and metadata files. This can be done whether by specifying the system's environment variables or adding it otherwise to the class path of the application, for example, specifying it with a start parameter if possible.
  - The alternative is to put the Jar file in the JRE/lib/ext folder, which is used by the Java application, so that it will be loaded by starting the JVM. Here can also be the driver deployed, if the application does not give the option to specify a jar file for the driver. This approach is usually not a good programming model, but it can be used if no other option is available.

Of course you can also ask the application provider to add official support for the IMS Universal drivers to his product by allowing to select the class path or jar files of the IMS Universal JDBC driver and the metadata file.

9.3 Tracing in problem cases

There are several situations where you might want enable tracing to find the problem cause, and we discuss those situations in this section.

9.3.1 IMS Universal driver tracing

The IMS Universal Drivers have several places where you can set the tracing option, depending on your runtime environment.

Tracing in JRE applications

One option for tracing in JRE applications is to turn on automatic tracing in the JRE logging.properties file.

You can set the trace level for the IMS Universal drivers loggers in the logging.properties file of your Java Runtime Environment (JRE). Using this method, the application does not need to be recompiled. The file is located on the install path of your JRE, under \jre\lib\logging.properties. To set the trace for all IMS Universal drivers loggers, add the lines in Example 9-2 on page 215 to the logging.properties file to send the trace output to a file.

Example 9-2 logging.properties entries for tracing

```
com.ibm.ims.db.opendb.level = FINEST
java.util.logging.FileHandler.level = FINEST
java.util.logging.FileHandler.pattern = c:/UniversalDriverTrace.txt
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
```

Note: Some JREs overwrite these settings, and you will find the log file in a different location. In the case of Rational Developer for System z with enabled logging, it puts the log file as .log file in the .metadata directory of your workspace directory.

Tracing in JEE environments

Tracing can be turned on from your JEE application server. In WebSphere Application Server, this is configured through the administrative console. The IMS Universal DB resource adapter must be deployed on WebSphere Application Server before tracing can be configured. To get the most detailed trace from the IMS Universal DB resource adapter:

1. Log on to the WebSphere Application Server administration console, and select **Troubleshooting** → **Logs and Trace**.
2. Select your **application server**.
3. Select **General Properties** → **Diagnostic Trace**.
4. Select **Additional Properties** → **Change Log Detail Levels**.
5. Switch to the **Runtime** tab.
6. Select the **Save runtime changes to configuration as well** option.
7. Select **Change Log Details Levels**, and choose the component **com.ibm.ims.db.opendb**.
8. In the Message and Trace levels menu, select the message level **FINEST**, and click **Apply**.
9. To save these changes, for the next time the application server is started, at the top of the page, click **Save**. WebSphere Application Server does not need to be restarted.

Tracing in applications

You can also programmatically turn on tracing in your IMS Universal driver application, which requires the application to be recompiled:

- Import the java.util.logging package in your application, and create a logger by calling the Logger.getLogger method with the String argument com.ibm.ims.db.opendb.
- In your application, you can set the level of tracing for the logger using the Logger.setLevel method.

The sample code in Example 9-3 shows how programmatic trace is enabled for any IMS Universal drivers application.

Example 9-3 Application enabled tracing

```
import java.util.logging;
...
private static final Logger imslog = Logger.getLogger("com.ibm.ims.db.opendb");
imslog.setLevel(Level.FINEST);
FileHandler fh = new FileHandler("C:/UniversalTrace.txt");
fh.setFormatter(new SimpleFormatter());
fh.setLevel(Level.FINEST);
imslog.addHandler(fh);
```

9.3.2 ODBM tracing

ODBM is the layer between IMS Connect and IMS. It is a Common Service Layer (CSL) address space and offers the usual tracing capabilities of CSL and Base Primitive Environment Tracing. Use this type of tracing to determine if it is a component error or not.

To start or stop a trace use the following commands:

```
UPDATE ODBM START(TRACE) DATASTORE(names)
UPDATE ODBM STOP(TRACE) DATASTORE(names)
```

For more information about CSL and BPE tracing visit the following web sites:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.sag/system_intro/ims_tracingbpecomponents.htm

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.dgr/ims_csl_service_aids.htm

9.3.3 IMS Tracing

If you receive an unexpected result for a query, it is a good idea to evaluate how your SQL call is described in the DL/I call format. For this purpose, you can use the tracing facility for DL/I calls with image captures using the IMS trace command. To take a DL/I call image capture (for example, for the PSB AUTPSB11), use the following sequence of events for tracing:

1. Turn on the trace with the following command:

   /TRACE SET ON PSB AUTPSB11 COMP

2. Run the Java application.
3. Turn off the trace with the following command:

   /TRACE SET OFF PSB AUTPSB11

4. Switch the online log data sets (OLDSs):

   /SWITCH OLDS

5. Execute the print utility DFSERA10, specifying the latest SLDS as input in SYSUT1 DD. The DFSERA10 input control statements to retrieve the information from the log look similar to the statements in Example 9-4.

Example 9-4 DFSERA10 options

```
OPTION PRINT OFFSET=5,VALUE=5F,COND=M
OPTION PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C,      X
VALUE=AUTPSB11,FLDLEN=7,DDNAME=OUTDDN,COND=E
```

9.4 Using tools with IMS Open Database

IBM IMS tools deliver the reliability and affordability you need to maximize the value of your IMS environment. Providing on demand access to IMS applications and data, the tools help optimize data across your enterprise. Information about IMS tools is available at:

http://www.ibm.com/software/data/db2imstools/products/ims-tools.html

Tools are grouped in solution packs for convenience. Of particular interest is the Performance Solution Pack, which is being extended to offer functionalities in the IMS Open Database area. The Performance Solution pack is described at:

http://www.ibm.com/software/data/db2imstools/imstools/ims-performance-solution-pack/

The IMS Performance Solution Pack (program number: 5655-S42) provides improved productivity for problem analysts, more efficient IMS application performance, improved IMS resource utilization, and higher system availability. It includes:

- IMS Connect Extensions help improve the availability, reliability, and performance of IMS Connect.
- IMS Performance Analyzer provides information about IMS system performance for monitoring, tuning, managing service levels, analyzing trends, and capacity planning.
- IMS Problem Investigator provides services that can help determine the cause of problems and trace the flow of events end-to-end.

We describe functions for Open Database provided by IMS Connect Extensions and IMS Problem Investigator.

9.4.1 IMS Connect Extensions

We know that IMS Connect, an integrated component of IMS since Version 9, provides TCP/IP access to IMS applications. With the introduction of Open Database, its role was extended to also provide the TCP/IP transport for direct access to IMS data. Its role is to transport and route DDM messages to the appropriate ODBM address space and return responses to clients through TCP/IP. It is a crucial component for distributed access with DRDA.

IBM IMS Connect Extensions for z/OS (IMS Connect Extensions) is an IMS Tool that provides event collection and instrumentation for IMS Connect. It collects information about outgoing and incoming requests that are transported using IMS Connect. This information is then available for analysis using standard IMS Tools, such as IMS Performance Analyzer and IMS Problem Investigator.

In the context of Open Database, we can divide the information that IMS Connect Extensions collects into two categories:

- Framing events
- Application-level events

Framing events

Framing events are events that mark significant points in the life cycle of an Open Database request. Each such event contains a time stamp that gives you information about the relative timings and duration of the entire DRDA request. These events include:

- Opening of the TCP/IP socket
- Processing of the DRDA request
- Security authentication and authorization
- Dispatch of request to ODBM address space
- Response received from ODBM address space
- Processing of response
- Dispatching to client
- Sync-point processing (if RRS is used)

Such events help you understand the flow of an Open Database request and obtain timings not only for the overall request but for individual processing steps for the request.

Application-level events

These events provide a record of exactly what request was sent by the client and what response was received back from IMS. They provide a complete breakdown of individual DDM objects and data. They help you debug and tune applications and audit and monitor exactly what activity is being performed by clients.

Figure 9-2 summarizes the value of this information.

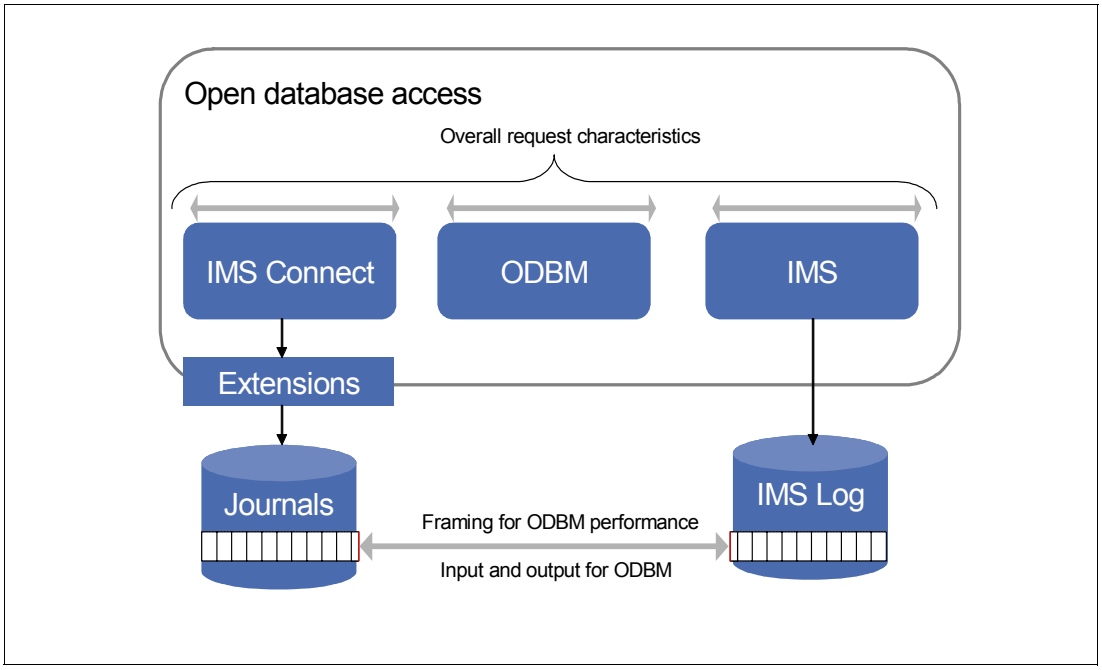


Figure 9-2 IMS Connect Extensions event collection

The information collected by IMS Connect Extensions can be analyzed using IMS Problem Investigator and IMS Performance Analyzer. Both tools can interpret the data and place it in the context of other information sources, such as the IMS log. IMS Problem Investigator focuses on giving you detailed insight into the characteristics of individual requests, while IMS Performance Analyzer focuses on providing you aggregate characteristics of Open Database activity so that you can study and analyze the performance of certain request types, server nodes, and so forth.

9.4.2 IMS Problem Investigator

Using the IBM IMS Problem Investigator for z/OS (IMS Problem Investigator) tool, you can analyze your IMS environment by interrogating IMS, CQS, Monitor logs, DB2 logs, and SMF data information collected by OMEGAMON ATF and TRF, as well as the information collected in IMS Connect Extensions journals. In the Open Database context, we focus on the analysis of the IMS Connect Extensions journals.

By allowing you to navigate, format, query, and extract these journals, you can better understand and gain insight into an Open Database request. IMS Problem Investigator can automatically identify records that are related to a request sequence from an Open Database client and display this information by tracking related log records.

In the example of Figure 9-3 on page 219, we track a sequence of Open Database requests. We can see framing events: the elapsed time between when the message was sent to ODBM, when a response was received, and information about the actual DDM objects.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
-------------------------------------------------------------------------------
BROWSE    CEX220.QAUNIT.EVNTLOG(WAS110)                     Tracking inactive
Command ===>                                                 Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2009-06-29 Monday     Time (Elapsed)
/  ---- ------------------------------------------------- ----------------
E  A049 READ Socket                                        09.52.05.464051
   A049 READ Socket                                               0.000095
   A05B DRDA 2001 ACCRDB-Access RDB                               0.000008
   A05D ODBM begin Allocate PSB (APSB) Program=AUTPSB11           0.000020
   A061 ODBM Routing Exit called                                  0.000010
   A062 ODBM Routing Exit returned                                0.000364
   A069 Message sent to ODBM                                      0.000945
   A06A Message received from ODBM                                0.430011
   A05E ODBM end Allocate PSB (DPSB) Program=AUTPSB11             0.000386
   A05C DRDA 2201 ACCRDBRM-Access RDB Reply Message               0.000025
   A04A WRITE Socket                                              0.000167
   A03C Prepare READ Socket                                       0.864489
   A049 READ Socket                                               0.000122
   A05B DRDA 200C OPNQRY-Open Query                               0.000009
   A049 READ Socket                                               0.000024
   A049 READ Socket                                               0.000026
   A05B DRDA CC05 DLIFUNC-DL/I function                           0.000008
```

Figure 9-3 Tracking a sequence of Open Database requests

If you are primarily interested in where the request spent its time, you can use this elapsed time view to identify how the requests are being processed. In Figure 9-3, the most significant time is spent waiting on a response from ODBM and IMS (0.43 seconds) and between the client's first request and second request (0.86 seconds). Most other aspects of the request, such as security authorization and processing within IMS Connect itself, took milliseconds.

The A05B and A05C log records contain the most relevant information from an application's standpoint. Figure 9-4 on page 220 shows how IMS Problem Investigator filtering allows you to view just these record types to give us the story of what the application was trying to do and how it was interacting with IMS itself.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
------------------------------------------------------------------------------
BROWSE    CEX000.QADATA.REDBOOK.DRDAT110.ICON.D1003  Record 00000043 More: < >
Command ===>                                                 Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2010-03-31 Wednesday  Time (LOCAL)
/  ---- ------------------------------------------------------ ----------------
   A049 READ Socket                                            13.21.47.348142
   A05B DRDA CC04 RTRVFLD-Field client wants to retrieve data  13.21.47.348149
   A049 READ Socket                                            13.21.47.348171
   A049 READ Socket                                            13.21.47.348194
   A05B DRDA CC06 SSALIST-List of segment search argument      13.21.47.348202
   A0AA ODBM Trace: Message sent to ODBM                       13.21.47.348617
   A069 Message sent to ODBM                                   13.21.47.348627
   A0AA ODBM Trace: Message received from ODBM                 13.21.47.350167
   A06A Message received from ODBM                             13.21.47.350178
   A05C DRDA 2205 OPNQRYRM-Open Query Complete                 13.21.47.350322
   A04A WRITE Socket                                           13.21.47.350496
   A048 Trigger Event for ODBMMSG                              13.21.47.350526
   A03C Prepare READ Socket                                    13.21.48.120400
   A049 READ Socket                                            13.21.48.120456
   A05B DRDA 2006 CNTQRY-Continue Query                        13.21.48.120464
   A0AA ODBM Trace: Message sent to ODBM                       13.21.48.120628
   A069 Message sent to ODBM                                   13.21.48.120684
```

Figure 9-4 Application records filtering

IMS Problem Investigator formats the standard DDM code points based on the DRDA specification and the IMS-specific code points. In Figure 9-4, we can see the initiation of the sync-point request, access request, the DL/I call, SSA list, and so forth. We can select these records to view more details of what they contain.

In Figure 9-5 on page 221, we can see the initiation of a request by an Open Database client.

```
File  Menu  Format  Help
------------------------------------------------------------------------------
BROWSE     CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000021 Line 00000019
Command ===>                                                 Scroll ===> CSR
Form    ===>          +     Use Form in Filter               Format ===> FORM
+001D  Type....... 01          RQSCRR..... 0001

+0020  Object..... 2001 ACCRDB-Access RDB
+0020  Length..... +60         CP......... 2001

+0024  Object..... 2110 RDBNAM-Relational Database Name
+0024  Length..... +17         CP......... 2110
+0028  Data....... 'AUTPSB11.ODB1'

+0035  Object..... 210F RDBACCCL-RDB Access Manager Class
+0035  Length..... +6          CP......... 210F        Data....... 2407

+003B  Object..... 112E PRDID-Product-specific Identifier
+003B  Length..... +20         CP......... 112E
+003F  Data....... 'IMS OPEN DB V1.0'

+004F  Object..... 002F TYPDEFNAM-Data Type Definition Name
+004F  Length..... +13         CP......... 002F
+0053  Data....... 'QTDSQL370'
```

Figure 9-5 Tracing a request initiation

You can also view segment search arguments (SSA), as shown in Figure 9-6.

```
File  Menu  Format  Help
------------------------------------------------------------------------------
BROWSE     CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000047 Line 00000015
Command ===>                                                 Scroll ===> CSR
Form    ===>          +     Use Form in Filter               Format ===> FORM
+0018  CERE_5B_VAR_CODEPOINT...... CC06

+001A  DSSHDR..... DSS header for DDM command
+001A  DSSlen..... +29         DDMID...... D0          FormatID... 03
+001D  Type....... 03          RQSCRR..... 0001

+0020  Object..... CC06 SSALIST-List of segment search argument
+0020  Length..... +23         CP......... CC06

+0024  Object..... C905 SSACOUNT-Number of segment search arguments
+0024  Length..... +6          CP......... C905        Data....... 0001

+002A  Object..... C906 SSA-Segment search argument
+002A  Length..... +13         CP......... C906
+002E  Data....... 'DEALER   '
******************************** End of data ********************************
```

Figure 9-6 Viewing segment search arguments

Importantly, you can also view exactly what the ODBM address space is sending as a response to the client, which allows you to isolate problems in parsing and interpreting output from IMS, as shown in Figure 9-7 on page 222.

```
File  Menu  Format  Help
------------------------------------------------------------------------------
BROWSE     CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000062 Line 00000019
Command ===>                                                 Scroll ===> CSR
Form    ===>          +     Use Form in Filter               Format ===> FORM
+001D  Type....... 03          RQSCRR..... 0001

+0020  Object..... 241B QRYDTA-Query Answer Set Data
+0020  Length..... +113        CP......... 241B
+0024  AIB........ aibStream
+0024  AIBflag.... 00          AIBused.... +61         AIBretc.... 00000000
+002D  AIBreas.... 00000000    AIBerrc.... 00000000
+0035  DBPCB...... dbpcbStream
+0035  DBPflag.... 00          DBflag..... 00
+0037  DBname..... 'AUTOLDB '  SL......... '01'        SC......... '  '
+0043  Segment.... 'DEALER  '  KFBAflag... 00          KFBAlen.... +4
+0050  KFBA....... '1234'
+0054  IOarea..... IO area
       +0000  F1F2F3F4 E2C1D540 D1D6E2C5 40C6D6D9   *1234SAN JOSE FOR*
       +0010  C4404040 40404040 40404040 40404040   *D               *
       +0020  4040E2C1 D540D1D6 E2C54040 F9F5F7F7   *  SAN JOSE  9577*
       +0030  F760F3F3 F3F3F7F7 F7F4F4F4 F4         *7-33337774444   *
********************************* End of data *********************************
```

Figure 9-7 ODBM response tracking

If you need to understand specific flags or fields, you can zoom on them to get more information, as shown in Figure 9-8.

```
+------------------------------- Field Zoom ---------------------------------+
|   File  Menu  Help                                                         |
| -------------------------------------------------------------------------- |
| BROWSE    CEX220.QAUNIT.EVNTLOG(WAS110)                    Line 00000000   |
| Command ===>                                             Scroll ===> CSR   |
| ****************************** Top of data ****************************** |
| +0024  AIBflag.... 00   AIB null indicator                                 |
|                                                                            |
|  On    Present.... 00   aibStream data structure is present                |
|  Off   Null....... FF   aibStream data structure contains no data after    |
|                         the AIB null indicator. The total length of the    |
|                         aibStream data structure is one byte.              |
| ****************************** End of data ****************************** |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
|                                                                            |
+----------------------------------------------------------------------------+
```

Figure 9-8 Zooming on specific fields

Other than browsing and drilling down on information that is specific to particular requests, you can also use filters to identify requests with particular characteristics, for example, identify requests arriving from a given server or for a given database.

9.4.3 Identifying and resolving problems

In this section, we show how you can use the IMS Tools to identify and resolve common problems with the sample applications. You can classify Open Database issues into the following categories:

- Session errors: Conditions that generate distinct errors, for example, specifying the wrong alias name or trying to access a stopped PSB.
- Performance problems: IMS provides an output, but processing time is slow.
- Unexpected responses: The client receives information from IMS, but it is not the feedback that the client was expecting.

Previously, we showed that IMS Problem Investigator can be used to examine the requests that the client sends, how they are processed by IMS, and the replies that are returned, allowing you to understand the context of unexpected responses. We now discuss session errors and performance problems.

Session errors

When a session error occurs, IMS Connect Extensions writes an A047 record to the journal. You can use IMS Problem Investigator's filtering to identify any session errors in the journal. Figure 9-9 shows a simple filter that shows only A047 records in the journal.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
-------------------------------------------------------------------------------
BROWSE    CEX000.QADATA.REDBOOK.ERR01.ICON.D100331           Tracking inactive
Command ===>                                                  Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2010-03-31 Wednesday  Time (LOCAL)
/  ---- ------------------------------------------------------ ----------------
   A047 Session Error                                          11.54.55.710442
   A047 Session Error                                          12.03.02.490039
   A047 Session Error                                          12.32.09.670281
****************************** Bottom of Data ******************************
```

Figure 9-9 Filtering by A047 records

The log record itself often contains enough information to understand the cause of the problem, for example, as shown in Figure 9-10 on page 224, if you select the last session error, you can see the message area, identifying that the client requested RRS when RRS was not available.

```
File  Menu  Format  Help
------------------------------------------------------------------------------
BROWSE     CEX000.QADATA.REDBOOK.ERR01.ICON.D100 Record 00000344 Line 00000010
Command ===>                                                 Scroll ===> CSR
Form    ===>          +     Use Form in Filter               Format ===> FORM
+000A  CERE_47_TASKID..... ID of task recording event
+000A  CERE_47_COL#....... 01  CERE_47_TKS#....... 04
+000C  CERE_47_EVKEY...... C5C21834F1A8FC61
+0014  CERE_47_VAR_LL..... 009C
+0016  CERE_47_VAR_APAR... 0001
+0018  CERE_47_VAR_FLAG3.......... 80
+001A  CERE_47_VAR_MSG.... 134 byte message area
       +0000  00640000 C8E6E2D2 F2F8F8F0 C540D9D9   *....HWSK2880E RR*
       +0010  E240C3D6 D4D4C1D5 C440C6C1 C9D3C5C4   *S COMMAND FAILED*
       +0020  5E40C37E D6C4C2F2 C5F4F2F3 6B40C3D7   *; C=ODB2E423, CP*
       +0030  7EE2E8D5 C3C3E3D3 4040406B 40D77EF4   *=SYNCCTL   , P=4*
       +0040  F8F8F5F5 4040406B 40D97EF0 F0F0F46B   *8855   , R=0004,*
       +0050  40D9E27E D9D9E2D5 C1E5C9D3 6B40D47E   * RS=RRSNAVIL, M=*
       +0060  D4D9C3E5 00000000 00000000 00000000   *MRCV............*
       +0070  00000000 00000000 00000000 00000000   *................*
       +0080  00000000 0000                         *......          *
+00A0  CERE_47_VAR_SESRSN......... 'WRITE   '
+00A8  CERE_47_VAR_TOKEN.......... 0000000000000000
******************************** End of data ********************************
```

Figure 9-10 Displaying the message area

In some cases, you might be interested in examining the complete flow of a request that generated a session error. In such cases, you can use tracking (TX line action) to reveal all of the event records that are associated with a particular session error , as shown in Figure 9-11 on page 225.

In this case, you can see the progression of the request leading up to the session error. The client successfully completes security authentication, but when the Access RDB request is made, triggering the allocation of the PSB, the request then receives an RDB Not Found DRDA object, which triggers the session error.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
------------------------------------------------------------------------------
BROWSE    CEX000.QADATA.REDBOOK.ERR01.ICON.D100331   Record 00000013 More: < >
Command ===>                                                 Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2010-03-31 Wednesday  Time (Relative)
/  ---- ------------------------------------------------------ ----------------
TX A049 READ Socket                                                  -0.256794
   A05B DRDA 106E SECCHK-Security Check                              -0.256786
   A063 ODBM Security Exit called                                    -0.256755
   A064 ODBM Security Exit returned                                  -0.256668
   A05C DRDA 1219 SECCHKRM-Security Check Reply Message              -0.256594
   A04A WRITE Socket                                                 -0.256516
   A049 READ Socket                                                  -0.000293
   A049 READ Socket                                                  -0.000223
   A05B DRDA 2001 ACCRDB-Access RDB                                  -0.000216
   A05D ODBM begin Allocate PSB (APSB) Program=AUTPSB11              -0.000194
   A061 ODBM Routing Exit called                                     -0.000185
   A062 ODBM Routing Exit returned                                   -0.000033
   A05C DRDA 2211 RDBNFNRM-RDB Not Found                          11.54.55.710366
   A04A WRITE Socket                                                 +0.000064
   A047 Session Error                                                +0.000075
   A00C Begin CLOSE Socket                                           +0.000108
   A00D End CLOSE Socket                                             +0.000325
```

Figure 9-11 Tracking the complete flow

Performance problems

Using IMS Problem Investigator, you can map relative timing of an Open Database request and broadly identify how much time the requests spend in various stages of processing:

- Client
- IMS Connect
- ODBM
- IMS

The log records that are available for IMS (often the area where critical insight is required) are similar to those that are available for CICS DBCTL transactions. As such, they provide only limited information about the timings of the DL/I calls themselves. If the requests are predominantly queries and not updates, very little information is actually available in the IMS logs.

To mitigate this, use database trace, such as the IMS Monitor. However, the most detailed trace is available from the OMEGAMON for IMS on z/OS application trace facility (ATF). Using ATF, you can record DL/I call characteristics. IMS Tools allow you to connect between the DL/I call activity and the DRDA requests that are being passed by the client, as shown in Figure 9-12 on page 226.

Figure 9-12 on page 226 shows the Open Database requests and the DL/I calls themselves, including the duration of each call and CPU utilization for each call.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
-------------------------------------------------------------------------------
BROWSE    CEX000.QADATA.REDBOOK.DRDAT111.ICON.D1003  Record 00000308 More: < >
Command ===>                                                  Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2010-03-31 Wednesday  Time (LOCAL)
/  ---- ------------------------------------------------------ ----------------
   A049 READ Socket                                            13.46.47.095038
   A05B DRDA CC06 SSALIST-List of segment search argument      13.46.47.095045
   A0AA ODBM Trace: Message sent to ODBM                       13.46.47.095985
   A069 Message sent to ODBM                                   13.46.47.096016
   06   OSAM IWAIT start TranCode=ODBA02CD Region=0003         13.46.47.142891
   20   Database Open Database=EMPDB2 Region=0003              13.46.47.143647
   06   OSAM IWAIT start TranCode=ODBA02CD Region=0003         13.46.47.181506
   20   Database Open Database=AUTODB Region=0003              13.46.47.182252
   06   OSAM IWAIT start TranCode=ODBA02CD Region=0003         13.46.47.191442
   01   DLI GHU Database=EMPLDB2 SC='  ' Elapse=0.095875       13.46.47.096570
   B021 DLI Database Trace Database=EMPLDB2 Func=GHU           13.46.47.192378
   A0AA ODBM Trace: Message received from ODBM                 13.46.47.192881
   A06A Message received from ODBM                             13.46.47.192909
   A05C DRDA 2205 OPNQRYRM-Open Query Complete                 13.46.47.193186
   A04A WRITE Socket                                           13.46.47.193515
   A048 Trigger Event for ODBMMSG                              13.46.47.193554
   A03C Prepare READ Socket                                    13.46.48.120636
```

Figure 9-12 DRDA and DL/I flow

You can control the amount of information IMS Problem Investigator shows (to view extended details, scroll right (F11)). As shown in Figure 9-13, the ATF records provide information about the CPU utilization and elapsed time for each DLI request.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
-------------------------------------------------------------------------------
BROWSE    CEX000.QADATA.REDBOOK.DRDAT111.ICON.D1003  Record 00000316 More: < >
Command ===>                                                  Scroll ===> CSR
   Forwards / Backwards . . 00.00.00.000100    Time of Day . . 16.46.22.845746
   Code Description                Date 2010-03-31 Wednesday  LSN
/  ---- ------------------------------------------------------ ----------------
   06   OSAM IWAIT start                                       2-00000000000004
        TranCode=ODBA02CD Program=AUTPSB11 Userid=AUTPSB11 Region=0003
        IMSID=ODBA02CD RecToken=ODBA02CD/0000000100000000 Elapse=0.000725
   -------------------------------------------------------------------------
   01   DLI GHU                                                2-00000000000005
        TranCode=ODBA02CD Program=AUTPSB11 Userid=AUTPSB11 Database=EMPLDB2
        Region=0003 IMSID=ODBA02CD RecToken=ODBA02CD/0000000100000000 SC='  '
        Elapse=0.095875 CPU=0.002190
   -------------------------------------------------------------------------
   B021 DLI Database Trace                                     3-00000000000105
        LTerm=EMPLDB2 Database=EMPLDB2 Region=0003
        RecToken=ODBA02CD/0000000100000000 Func=GHU Elapsed=00.095631
   -------------------------------------------------------------------------
   A0AA ODBM Trace: Message received from ODBM                 1-00000000000359
        IMSID=IBDEODOD LogToken=C5C228E17B387162
   -------------------------------------------------------------------------
   A06A Message received from ODBM                             1-0000000000035A
```

Figure 9-13 Displaying detailed information with F11

IMS Problem Investigator correlates the ATF data with the Open Database request itself so that you can see exactly which request led to which DLI calls and what their timings were.

As shown in Figure 9-14, if you select the ATF generated 01 record, you can see the IO area that is used for the request itself.

```
File  Menu  Format  Help
-------------------------------------------------------------------------------
BROWSE     CEX000.QADATA.REDBOOK.DRDAT111.ICON.D Record 00000317 Line 00000042
Command ===>                                                  Scroll ===> CSR
Form    ===>          +     Use Form in Filter                Format ===> FORM
+00DC  ATRDXEL.... DL/I Trace Element
+00DC  ATRDX@E.... 14ABD400    ATRDXTY.... 02          ATRDXF..... 00
+00E2  ATRDX#..... 0008
+00E4  ATRDXV..... Element Data - Key Feedback Area
       +0000  F2F2F2F2 F2F2C3C1                     *222222CA        *

+00EC  ATRDXEL.... DL/I Trace Element
+00EC  ATRDX@E.... 14AAD2E8    ATRDXTY.... 01          ATRDXF..... 00
+00F2  ATRDX#..... 004C
+00F4  ATRDXV..... Element Data - I/O Area
       +0000  F2F2F2F2 F2F2C299 96A69540 40404040   *222222Brown     *
       +0010  40404040 40404040 40404040 404040D9   *               R*
       +0020  96954040 40404040 40404040 40404040   *on              *
       +0030  40404040 40404040 E2D6D4C5 40E2E3D9   *         SOME STR*
       +0040  C5C5E340 40404040 40404040            *EET             *

+0140  ATRDXEL.... DL/I Trace Element
+0140  ATRDX@E.... 14AAD540    ATRDXTY.... 04          ATRDXF..... 00
+0146  ATRDX#..... 0036
```

Figure 9-14 Displaying the I/O area

9.5 Additional sample programs

If you need more examples for your understanding of using the IMS Universal DB drivers in your application, there are several self-explaining samples for download on the web.

If you want to use the CSLDMI interface for writing an application, there is a full description of the CSLDMI interface in section *IMS Version 11 System Programming APIs*, SC19-2445, and an example of using this is in section “C.2 Sample ODBM program” of *IMS Version 11 Technical Overview,* SG24-7807.

If you want to use a REXX script for accessing IMS Connect with Open Database, refer to *IMS Version 11 Technical Overview,* SG24-7807.

In addition to the examples that we use in this book, we provide some more samples as additional material. For information about how to download the samples see Appendix D, “Additional material” on page 251.

A

IBM DB2 Data Server drivers and clients

This appendix provides an overview of the DB2 drivers that you can use to access DB2 data from the same client Java application using IMS drivers to use DL/I data.

For more information, refer to *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01, which is the base for this appendix.

The IBM Data Server driver for JDBC and SQLJ (formerly known as IBM DB2 driver for JDBC and SQLJ) is a single application driver to support the most demanding Java applications. This driver can be used as in type-4 or type-2 mode, includes support for pureXML®, SQL/XML, and XQuery, and it is optimized for DB2 across all platforms (Linux, UNIX, Windows, z/OS and iSeries®) and Informix Dynamic Server (IDS).

A.1 IBM Data Server drivers and clients

IBM delivered a variety of client products for application developers and database administrators to support distributed access to data stored in DB2 for z/OS.

Here is the list of the IBM Data Server clients and drivers:

- IBM Data Server Client
- IBM Data Server Runtime Client
- IBM Data Server driver for ODBC and CLI
- IBM Data Server driver for JDBC and SQLJ
- IBM Data Server Driver Package

In the IMS scenario in this book, we use the IBM Data Server driver for JDBC and SQLJ, but you can use any of the drivers that support JDBC (see Appendix A.1.6, “Driver and client comparison” on page 233). To get the driver, you need the driver itself, which you can download using your IBM ID from the following web site:

http://www.ibm.com/support/docview.wss?rs=4020&uid=swg21385217

Additionally, if you want to connect to DB2 on z/OS, you need the correct license jar file.

In this appendix we briefly describe these drivers and clients and provide a table that compares the driver and client products. Refer to the IBM DB2 9.7 Information Center at the following web site for more information about these products:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.swg.im.dbclient.install.doc/doc/c0022612.html

A.1.1 IBM Data Server driver for JDBC and SQLJ

JDBC is an application programming interface (API) that Java applications use to access relational databases. SQLJ provides support for embedded static SQL in Java applications. In general, Java applications use JDBC for dynamic SQL and SQLJ for static SQL.

This driver is also called the JAVA Common Client (JCC) driver and was formerly known as the IBM DB2 Universal Database driver. The DB2 JDBC type-2 driver for LUW, also called the CLI legacy driver, is deprecated.

We can refer to these clients as Java-based clients.

See the Java application development for IBM data servers section at the DB2 Version 9.5 for Linux, UNIX, and Windows Information Center:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0024189.html

DB2 support for JDBC drivers

The IBM Data Server driver for JDBC and SQLJ provides type-4 and type-2 connectivity. To communicate with remote servers using DRDA, the type-4 driver is used as a DRDA Application Requester:

- Driver for JDBC type-2 connectivity (type-2 driver)

  Type-2 drivers are written partly in the Java programming language and partly in native code. The drivers use a native client library that is specific to the data source to which they connect. Because of the native code, their portability is limited. Use of the type-2 driver to

connect to DB2 for z/OS is recommended for WebSphere Application Server running on System z.

- Driver for JDBC type-4 connectivity (type-4 driver)

  Type-4 drivers are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source. The JDBC type-4 driver is recommended to connect distributed Java applications to DB2 for z/OS data.

  IBM ships two versions of the JDBC type-4 driver with the IBM Data Server driver for JDBC and SQLJ V9.5 FP3 product:

  – Version 3.5x is JDBC 3.0-compliant. It is packaged as db2jcc.jar and sqlj.zip and provides JDBC 3.0 and earlier support.

  – Version 4.x is JDBC 3.0-compliant and supports some JDBC 4.0 functions. It is packaged as db2jcc4.jar and sqlj4.zip.

  The type-4 driver provides support for distributed transaction management. This support implements the Java Platform, Enterprise Edition (JEE), Java Transaction Service (JTS), and Java Transaction API (JTA) specifications, which conform to the X/Open standard for distributed transactions. Distributed Transaction Processing: The XA Specification, is available at:

  http://www.opengroup.org

Applications that use the IBM Data Server driver for JDBC and SQLJ to access DB2 for z/OS data across a network implement the type-4 driver. In Figure A-1, an application uses the IBM Data Server driver for JDBC and SQLJ to access a stand-alone DB2.

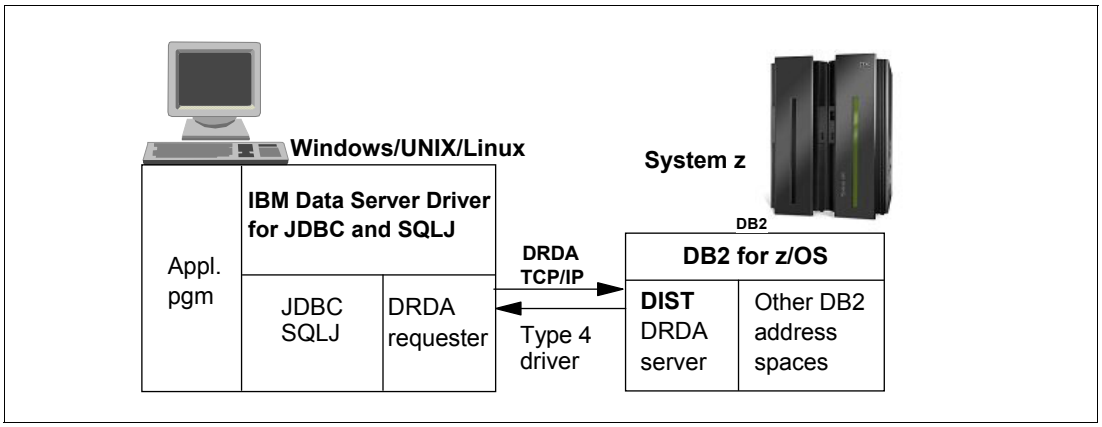


Figure A-1 IBM Data Server driver for JDBC and SQLJ connecting directly to DB2 for z/OS

In the remainder of the discussion and examples of the IBM Data Server driver for JDBC and SQLJ, we refer to the type-4 driver support.

A.1.2 IBM Data Server driver for ODBC and CLI

This product is for applications using ODBC or CLI only and provides a lightweight deployment solution designed for ISV deployments. This driver, also referred to as CLI driver, provides runtime support for applications using the ODBC API or CLI API, without the need of installing the IBM Data Server Client or the IBM Data Server Runtime Client.

The CLI driver is conceptually similar to the JDBC type-4 driver. The CLI driver is packaged in a small footprint, providing the DRDA AR functions necessary to connect to DB2 for z/OS for those application scenarios where you do not require robust tools, development, or administration functions.

A.1.3 IBM Data Server Driver Package

IBM Data Server Driver Package provides a lightweight deployment solution, providing runtime support for applications using ODBC, CLI, .NET, OLE DB, open source, or Java APIs without needing to install Data Server Runtime Client or Data Server Client. This driver has a small footprint and is designed to be redistributed by independent software vendors (ISVs) and to be used for application distribution in mass deployment scenarios typical of large enterprises.

The IBM Data Server Driver Package capabilities include:

- Support for applications that use ODBC, CLI, or open source (PHP or Ruby) to access databases.
- Support for client applications and applets that are written in Java using JDBC and for embedded SQL for Java (SQLJ).
- IBM Informix Dynamic Server support for .NET, PHP, and Ruby.
- Application header files to rebuild the open source drivers.
- Support for DB2 Interactive Call Level Interface (db2cli).
- On Windows operating systems, IBM Data Server Driver Package also provides support for applications that use .NET or OLE DB to access databases. In addition, this driver is available as an installable image, and a merge module is available to allow you to easily embed the driver in a Windows Installer-based installation.
- On Linux and UNIX operating systems, IBM Data Server Driver Package is not available as an installable image.

A.1.4 IBM Data Server Runtime Client

Using this product, you can run applications on remote databases. Graphical user interface (GUI) tools are not included. The capabilities are:

- Command line processor (CLP)
- Base client support for database connections, SQL statements, XQuery statements and commands
- Support for common database access interfaces (JDBC, SQLJ, ADO.NET, OLE DB, ODBC, command line interface (CLI), PHP and Ruby), including drivers and ability to define data sources
- Lightweight Directory Access Protocol (LDAP) exploitation
- Support for TCP/IP and Named Pipe
- Support for multiple concurrent copies and various licensing and packaging options

The IBM Data Server Runtime Client (Runtime Client) has a rich set of SQL APIs for deployment in more complex application environments.

A.1.5 IBM Data Server Client

This is the full-function product for application development, database administration, and client/server configuration. Its capabilities are:

- Configuration Assistant
- Control Center and other graphical tools
- First Steps for new users

- Visual Studio tools
- IBM Data Studio
- Application header files
- Precompilers for various programming languages
- Bind support
- All The functions included in the IBM Data Server Runtime Client

A.1.6 Driver and client comparison

Table A-1 summarizes the highlights of the IBM Data Server products. Refer to standard DB2 for LUW product documentation for additional details.

Table A-1 IBM Data Server drivers and clients comparison

Product	Smallest footprint	JDBC and SQLJ	ODBC and CLI	OLE DB and .NET	Open Source	CLP	DBA, Dev, GUI tools
IBM Data Server driver for JDBC and SQLJ	X	X					
IBM Data Server driver for ODBC and CLI	X		X				
IBM Data Server Driver Package		X	X	X	X		
IBM Data Server Runtime Client		X	X	X	X	X	
IBM Data Server Client		X	X	X	X	X	X

For the correct version of the driver, visit the following web site:
http://www.ibm.com/support/docview.wss?rs=71&uid=swg21363866

A.2 Support for JDBC and SQLJ

To develop a Java application from your client, you need the appropriate level of IBM Software Development Kit (SDK) for Java or DB2 for Linux, UNIX, and Windows. This is true to use Java-based tools and to create and run Java applications, including stored procedures and user-defined functions.

If the IBM SDK for Java is required by a component being installed and the SDK for Java is not already installed in that path, the SDK for Java is installed if you use either the DB2 Setup wizard or a response file to install the product.

Refer to the Application Development with DB2 web site for details:

http://www.ibm.com/software/data/db2/ad/java.html

If you need to provide support for JDBC and SQLJ requesters for the first time, there are several steps you must complete. Most of these steps are for DB2 for z/OS as the DRDA AS, but some relate to the requesters.

To install support for JDBC and SQLJ:

1. Allocate and load IBM Data Server driver for JDBC and SQLJ libraries. You perform this step on the client(s).
2. On DB2 for z/OS, set the DESCSTAT parameter to YES (DESCRIBE FOR STATIC on the DSNTIPF installation panel). This is necessary for SQLJ support.
3. In z/OS UNIX System Services, edit the .profile file to customize environment variable settings.
4. Optional: Customize IBM Data Server driver for JDBC and SQLJ configuration properties. This refers to the properties on the clients.
5. On DB2 for z/OS, enable the DB2-supplied stored procedures, and define the tables that the IBM Data Server driver for JDBC and SQLJ uses.
6. In z/OS UNIX System Services, run the DB2Binder utility to bind the packages for the IBM Data Server driver for JDBC and SQLJ.
7. Install z/OS Application Connectivity to DB2 for z/OS feature. This applies if you have a JDBC or SQLJ application on z/OS in an LPAR where you do not have a DB2 for z/OS subsystem. This feature is effectively the type-4 driver for z/OS and provides the DRDA AR function without a local DB2 (and DDF) to communicate with a DB2 for z/OS server elsewhere in the network.

Refer to *DB2 Version 9.1 for z/OS Installation Guide,* GC18-9846, for more information about these steps.

A.3 Using the IBM Data Server driver for JDBC and SQLJ

We focus on the type-4 driver in this section because it can be used to connect to a DB2 for z/OS server through DRDA. The type-4 driver is now delivered in the jcc3 and jcc4 streams. The jcc4 stream requires JDK 1.6 to be installed.

You also must customize and run the DSNITJMS job that creates stored procedures and tables that the type-4 driver requires. Refer to the following links for complete information:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.java.doc/doc/t0024156.html

http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0052041.html

To know which version of the driver you are using, use the command in Example A-1 when your driver is in your Windows class path.

Example A-1 Determining the driver version

```
C:\DDF\test>java com.ibm.db2.jcc.DB2Jcc -version
IBM Data Server Driver for JDBC and SQLJ 4.8.23
```

Connecting to a DB2 for z/OS server using the type-4 driver

You can use either the DriverManager or the DataSource interface to obtain a connection to the database server. Here is an example using the DriverManager.getConnection() interface where the connection properties are embedded in the application program. Example A-2 on page 235 shows the use of the getConnection() interface to obtain a connection to the DB2 for z/OS server.

Example A-2 Using the getConnection()

```
String url = "jdbc:db2://wtsc63.itso.ibm.com:12347/DB9A
java.util.Properties prop = new  java.util.Properties();
prop.put("user", user);
prop.put("password", password);
prop.put("driverType", "4");
conn1 = DriverManager.getConnection(url,prop);
```

It is possible to create and use a DataSource object in the same application similar to the DriverManager interface; however, this method does not provide portability. The recommended way to use a DataSource object is for your system administrator to create and manage it separately using WebSphere Application Server or some other tool. It is then possible for your system administrator to modify the data source attributes, and you do not need to change your application program. Example A-3 shows the use of the DataSource interface to obtain a connection to the DB2 for z/OS server.

Example A-3 Connecting to DB2 for z/OS through the DataSource interface

```
DB2SimpleDataSource dataSource = new com.ibm.db2.jcc.DB2SimpleDataSource();
dataSource.setServerName (servername);
dataSource.setPortNumber (Integer.parseInt(port));
dataSource.setDatabaseName (databasename);
dataSource.setUser (user);
dataSource.setPassword (password);
dataSource.setDriverType (4);
conn1 = dataSource.getConnection();
```

For details about connections using the DriverManager or DataSource interfaces, refer to:

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/cjvjdcon.html

To learn more about using WebSphere to deploy DataSource objects, visit:

http://www.ibm.com/software/webservers/appserv/

In general, the default properties that are enabled for the type-4 driver are designed to maximize distributed performance. If you want to change either the configuration properties that have driver-wide scope or Connection/Data source properties that are application-specific, refer to the following web site.

http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/rjvdsprp.html

B

Car Dealer IVP database

The Car Dealer database is used in most of the examples in this book. This database is part of the Installation Verification Procedure (IVP) of IMS. It is readily available in your IMS environments, usually in test environments rather than production environments.

B.1 Car Dealer database overview

The Car Dealer database uses mainly two physical databases (autos and employees) and two indexes. The database builds a logical database across these two databases. We do not use the indexes in our example so we skipped them in the following sources and overviews.

B.1.1 AUTOLPCB overview diagram

Figure B-1 provides an overview diagram of the AUTOLPCB.

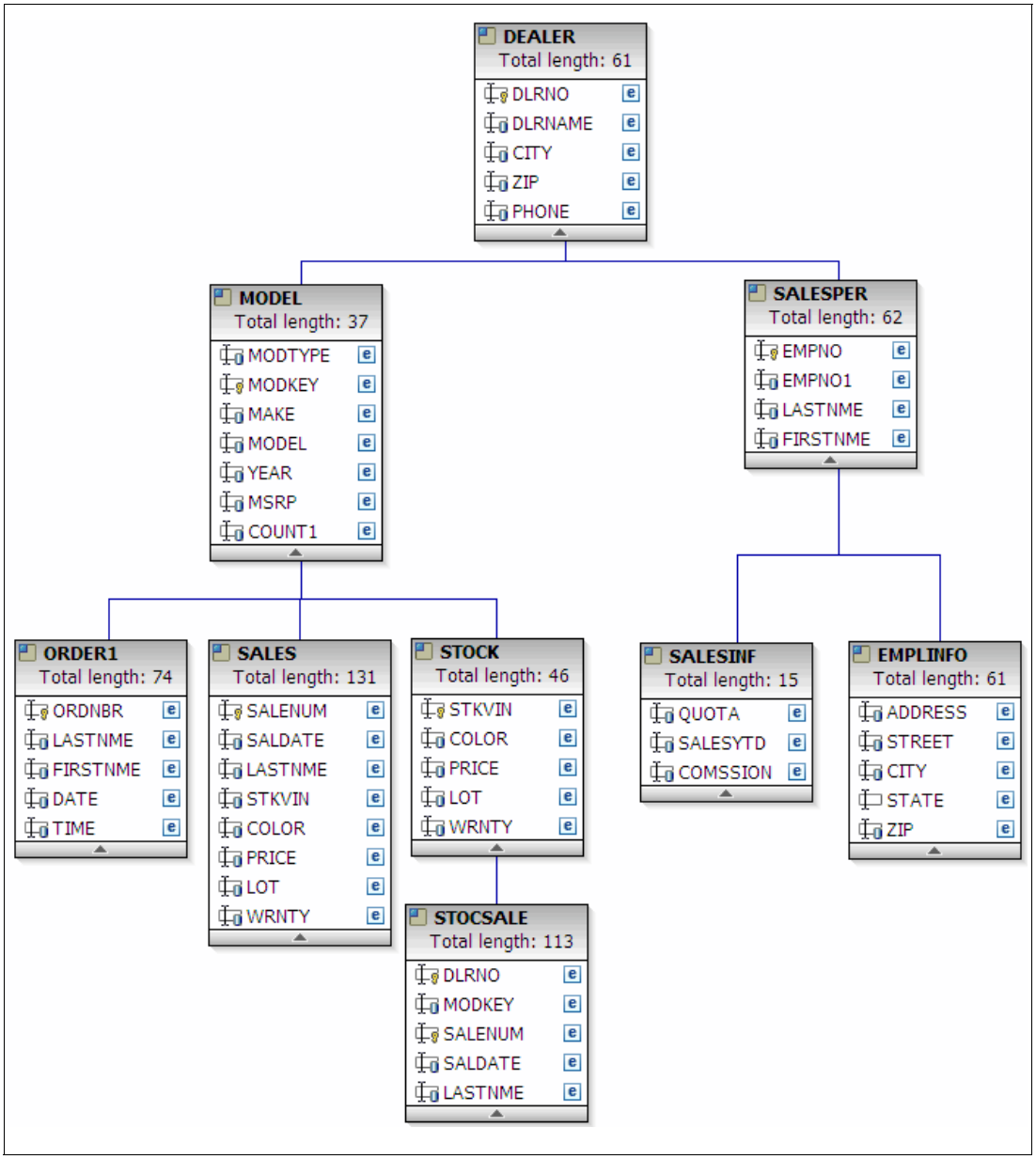


Figure B-1 AUTOLPCB overview diagram

B.1.2 EMPLPCB overview diagram

Figure B-2 on page 239 provides an overview diagram of the EMPLPCB.

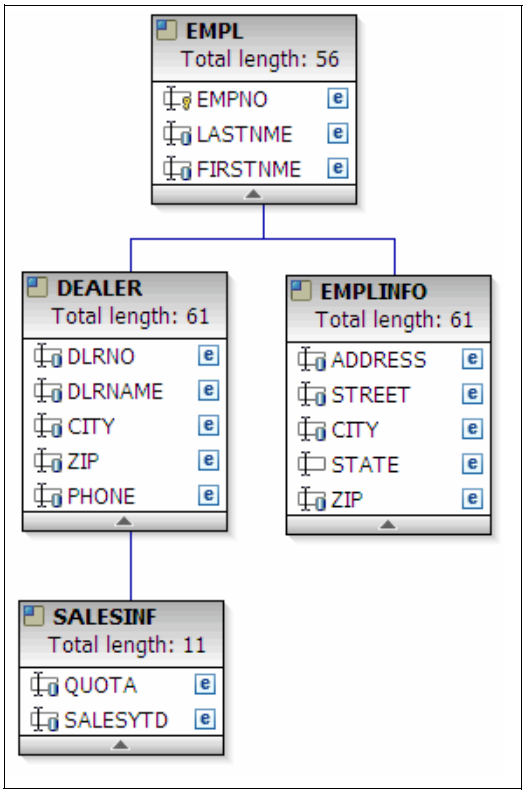


Figure B-2 EMPLPCB overview diagram

B.1.3 Metadata description

Example B-1 shows the used segments and fields of the database unmodified as CHAR fields.

Example B-1 DLIModel IMS Java Report

```
Class: AUTPSB11DatabaseView  in package: samples.ims.openDb  generated for PSB: AUTPSB11
================================================
PCB: AUTOLPCB
================================================
Segment: DEALER
  Field: DLRNO     Type=CHARLength=4++ Primary Key Field ++
  Field: DLRNAME   Type=CHARLength=30     (Search Field)
  Field: CITY      Type=CHARLength=10     (Search Field)
  Field: ZIP       Type=CHARLength=10     (Search Field)
  Field: PHONE     Type=CHARLength=7     (Search Field)
================================================
    Segment: MODEL
      Field: MODTYPE   Type=CHARLength=2     (Search Field)
      Field: MODKEY    Type=CHARLength=24++ Primary Key Field ++
      Field: MAKE      Type=CHARLength=10     (Search Field)
      Field: MODEL     Type=CHARLength=10     (Search Field)
      Field: YEAR      Type=CHARLength=4     (Search Field)
      Field: MSRP      Type=CHARLength=5     (Search Field)
      Field: COUNT1    Type=CHARLength=2     (Search Field)
    ================================================
        Segment: ORDER1
          Field: ORDNBR    Type=CHARLength=6++ Primary Key Field ++
          Field: LASTNME   Type=CHARLength=25     (Search Field)
```

```
          Field: FIRSTNME  Type=CHARLength=25     (Search Field)
          Field: DATE      Type=CHARLength=10     (Search Field)
          Field: TIME      Type=CHARLength=8     (Search Field)
        ================================================
        Segment: SALES
          Field: SALENUM   Type=CHARLength=4++ Primary Key Field ++
          Field: SALDATE   Type=CHARLength=8     (Search Field)
          Field: LASTNME   Type=CHARLength=25     (Search Field)
          Field: STKVIN    Type=CHARLength=20     (Search Field)
          Field: COLOR     Type=CHARLength=10     (Search Field)
          Field: PRICE     Type=CHARLength=5     (Search Field)
          Field: LOT       Type=CHARLength=10     (Search Field)
          Field: WRNTY     Type=CHARLength=1     (Search Field)
        ================================================
        Segment: STOCK
          Field: STKVIN    Type=CHARLength=20++ Primary Key Field ++
          Field: COLOR     Type=CHARLength=10     (Search Field)
          Field: PRICE     Type=CHARLength=5     (Search Field)
          Field: LOT       Type=CHARLength=10     (Search Field)
          Field: WRNTY     Type=CHARLength=1     (Search Field)
        ================================================
            Segment: STOCSALE
              Field: DLRNO     Type=CHARLength=4++ Primary Key Field ++
              Field: MODKEY    Type=CHARLength=24     (Search Field)
              Field: SALENUM   Type=CHARLength=4++ Primary Key Field ++
              Field: SALDATE   Type=CHARLength=8     (Search Field)
              Field: LASTNME   Type=CHARLength=25     (Search Field)
    ================================================
    Segment: SALESPER
      Field: EMPNO     Type=CHARLength=6++ Primary Key Field ++
      Field: EMPNO1    Type=CHARLength=6     (Search Field)
      Field: LASTNME   Type=CHARLength=25     (Search Field)
      Field: FIRSTNME  Type=CHARLength=25     (Search Field)
    ================================================
        Segment: SALESINF
          Field: QUOTA     Type=CHARLength=5     (Search Field)
          Field: SALESYTD  Type=CHARLength=5     (Search Field)
          Field: COMSSION  Type=CHARLength=5     (Search Field)
        ================================================
        Segment: EMPLINFO
          Field: ADDRESS   Type=CHARLength=61     (Search Field)
          Field: STREET    Type=CHARLength=25     (Search Field)
          Field: CITY      Type=CHARLength=25     (Search Field)
          Field: STATE     Type=CHARLength=2++ Primary Key Field ++
          Field: ZIP       Type=CHARLength=9     (Search Field)
================================================
PCB: EMPLPCB
================================================
Segment: EMPL
  Field: EMPNO     Type=CHARLength=6++ Primary Key Field ++
  Field: LASTNME   Type=CHARLength=25     (Search Field)
  Field: FIRSTNME  Type=CHARLength=25     (Search Field)
================================================
    Segment: DEALER
      Field: DLRNO     Type=CHARLength=4     (Search Field)
      Field: DLRNAME   Type=CHARLength=30     (Search Field)
      Field: CITY      Type=CHARLength=10     (Search Field)
      Field: ZIP       Type=CHARLength=10     (Search Field)
      Field: PHONE     Type=CHARLength=7     (Search Field)
    ================================================
```

```
      Segment: SALESINF
        Field: QUOTA     Type=CHARLength=5     (Search Field)
        Field: SALESYTD  Type=CHARLength=5     (Search Field)
    ================================================
    Segment: EMPLINFO
      Field: ADDRESS   Type=CHARLength=61     (Search Field)
      Field: STREET    Type=CHARLength=25     (Search Field)
      Field: CITY      Type=CHARLength=25     (Search Field)
      Field: STATE     Type=CHARLength=2++ Primary Key Field ++
      Field: ZIP       Type=CHARLength=9     (Search Field)
```

B.2 Car Dealer database source files

This section contains the source files of the Car Dealer IVP database for your reference.

B.2.1 AUTPSB11.psb

Example B-2 lists the source of the PSB, AUTPSB11 that we used in our scenarios.

Example B-2 AUTPSB11 PSB source

```
AUTOLPCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=AP,KEYLEN=100
         SENSEG NAME=DEALER,PARENT=0
         SENSEG NAME=MODEL,PARENT=DEALER
         SENSEG NAME=ORDER,PARENT=MODEL
         SENSEG NAME=SALES,PARENT=MODEL
         SENSEG NAME=STOCK,PARENT=MODEL
         SENSEG NAME=STOCSALE,PARENT=STOCK
         SENSEG NAME=SALESPER,PARENT=DEALER
         SENSEG NAME=SALESINF,PARENT=SALESPER
         SENSEG NAME=EMPLINFO,PARENT=SALESPER
AUTS1PCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=GRP,KEYLEN=100,           X
               PROCSEQ=SINDEX11
         SENSEG NAME=ORDER,PARENT=0
         SENSEG NAME=MODEL,PARENT=ORDER
         SENSEG NAME=DEALER,PARENT=MODEL
         SENSEG NAME=STOCK,PARENT=MODEL
*
AUTS2PCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=GRP,KEYLEN=64,            X
               PROCSEQ=SINDEX22
         SENSEG NAME=DEALER,PARENT=0
         SENSEG NAME=MODEL,PARENT=DEALER
         SENSEG NAME=STOCK,PARENT=MODEL
*
AUSI2PCB  PCB TYPE=DB,DBDNAME=SINDEX22,PROCOPT=GRDP,KEYLEN=28
          SENSEG NAME=SINDXB,PARENT=0

EMPLPCB   PCB TYPE=DB,DBDNAME=EMPLDB2,PROCOPT=AP,KEYLEN=10
          SENSEG NAME=EMPL,PARENT=0
          SENSEG NAME=DEALER,PARENT=EMPL
          SENSEG NAME=SALESINF,PARENT=DEALER
          SENFLD NAME=QUOTA,START=1
          SENFLD NAME=SALESYTD,START=7
          SENSEG NAME=EMPLINFO,PARENT=EMPL
```

```
          PSBGEN PSBNAME=AUTPSB11,LANG=JAVA
          END
```

B.2.2 AUTODB.dbd

Example B-3 shows the source of the physical DBD, AUTODB that is needed for use with PSB AUTPSB11.

Example B-3 AUTODB DBD source

```
DBD   NAME=AUTODB,ACCESS=(HDAM,OSAM),                          X
            RMNAME=(DFSHDC40,1,5,200)
         DATASET DD1=DFSDLR
         SEGM  NAME=DEALER,PARENT=0,BYTES=61
         FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
         FIELD NAME=DLRNAME,BYTES=30,START=5,TYPE=C
         FIELD NAME=CITY,BYTES=10,START=35,TYPE=C
         FIELD NAME=ZIP,BYTES=10,START=45,TYPE=C
         FIELD NAME=PHONE,BYTES=7,START=55,TYPE=C
         LCHILD  NAME=(SINDXB,SINDEX22),POINTER=INDX
         XDFLD NAME=XFLD2,SEGMENT=MODEL,                                X
               SRCH=(MAKE,MODEL),                                       X
               SUBSEQ=(YEAR,/SX1),                                      X
               DDATA=COUNT
         SEGM  NAME=MODEL,PARENT=DEALER,BYTES=37
         FIELD NAME=(MODKEY,SEQ,U),BYTES=24,START=3,                    X
               TYPE=C
         FIELD NAME=MODTYPE,BYTES=2,START=1,TYPE=C
         FIELD NAME=MAKE,BYTES=10,START=3,TYPE=C
         FIELD NAME=MODEL,BYTES=10,START=13,TYPE=C
         FIELD NAME=YEAR,BYTES=4,START=23,TYPE=C
         FIELD NAME=MSRP,BYTES=5,START=27,TYPE=P
         FIELD NAME=COUNT,BYTES=2,START=32,TYPE=P
         FIELD NAME=/SX1
         SEGM  NAME=ORDER,PARENT=MODEL,BYTES=74
         FIELD NAME=(ORDNBR,SEQ,U),BYTES=6,START=1,TYPE=C
         FIELD NAME=LASTNME,BYTES=25,START=7,TYPE=C
         FIELD NAME=FIRSTNME,BYTES=25,START=32,TYPE=C
         FIELD NAME=DATE,BYTES=10,START=57,TYPE=C
         FIELD NAME=TIME,BYTES=8,START=67,TYPE=C
         LCHILD  NAME=(SINDXA,SINDEX11),POINTER=INDX
         XDFLD NAME=XFLD1,SRCH=(LASTNME,FIRSTNME,ORDNBR),               X
               DDATA=DATE
         SEGM  NAME=SALES,PARENT=((MODEL,),(STOCK,PHYSICAL,AUTODB)),    X
               BYTES=85,                                                X
               POINTER=(LPARNT,LTWINBWD,TWINBWD),                       X
               RULES=(VVV)
         FIELD NAME=(SALENUM,SEQ,U),BYTES=4,START=49,TYPE=C
         FIELD NAME=SALDATE,BYTES=8,START=53,TYPE=C
         FIELD NAME=LASTNME,BYTES=25,START=61,TYPE=C
         SEGM  NAME=STOCK,PARENT=MODEL,BYTES=46
         LCHILD  NAME=(SALES,AUTODB),PAIR=STOCSALE,PTR=DBLE
         FIELD NAME=(STKVIN,SEQ,U),BYTES=20,START=1,TYPE=C
         FIELD NAME=COLOR,BYTES=10,START=21,TYPE=C
         FIELD NAME=PRICE,BYTES=5,START=31,TYPE=C
```

```
         FIELD NAME=LOT,BYTES=10,START=36,TYPE=C
         FIELD NAME=WRNTY,BYTES=1,START=46,TYPE=X
         SEGM  NAME=STOCSALE,PARENT=STOCK,PTR=PAIRED,                  X
               SOURCE=((SALES,DATA,AUTODB))
         FIELD NAME=(SALENUM,SEQ,U),BYTES=4,START=29,TYPE=C
         FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
         FIELD NAME=MODKEY,BYTES=24,START=5,TYPE=C
         FIELD NAME=SALDATE,BYTES=8,START=33,TYPE=C
         FIELD NAME=LASTNME,BYTES=25,START=41,TYPE=C
         SEGM  NAME=SALESPER,                                          X
               PARENT=((DEALER,),(EMPL,PHYSICAL,EMPDB2)),              X
               BYTES=6,                                                X
               POINTER=(LPARNT,LTWINBWD,TWINBWD),                      X
               RULES=(VVV)
         FIELD NAME=(EMPNO,SEQ,U),BYTES=6,START=1,TYPE=C
         SEGM  NAME=SALESINF,PARENT=SALESPER,BYTES=15
         FIELD NAME=QUOTA,BYTES=5,START=1,TYPE=C
         FIELD NAME=SALESYTD,BYTES=5,START=6,TYPE=C
         FIELD NAME=COMSSION,BYTES=5,START=11,TYPE=C
         DBDGEN
```

B.2.3 EMPDB2.dbd

Example B-4 shows the source of the physical DBD EMPDB2 that is needed for PSB AUTPSB11.

Example B-4 EMPDB2 DBD source

```
DBD    NAME=EMPDB2,ACCESS=(HDAM,OSAM),                          X
              RMNAME=(DFSHDC40,1,5,200)
        DATASET  DD1=DFSEMPL
        SEGM   NAME=EMPL,PARENT=0,BYTES=56
        LCHILD NAME=(SALESPER,AUTODB),PAIR=EMPSAL,POINTER=DBLE
        FIELD  NAME=(EMPNO,SEQ,U),BYTES=6,START=1,TYPE=C
        FIELD  NAME=LASTNME,BYTES=25,START=7,TYPE=C
        FIELD  NAME=FIRSTNME,BYTES=25,START=32,TYPE=C
        SEGM   NAME=EMPSAL,PARENT=EMPL,PTR=PAIRED,                 X
               SOURCE=((SALESPER,DATA,AUTODB))
        FIELD  NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
        SEGM   NAME=EMPLINFO,PARENT=EMPL,BYTES=61
        FIELD  NAME=(STATE,SEQ,M),BYTES=2,START=51,TYPE=C
        FIELD  NAME=ADDRESS,BYTES=61,START=1,TYPE=C
        FIELD  NAME=STREET,BYTES=25,START=1,TYPE=C
        FIELD  NAME=CITY,BYTES=25,START=26,TYPE=C
        FIELD  NAME=ZIP,BYTES=9,START=53,TYPE=C
       DBDGEN
       FINISH
       END
```

B.2.4 SINDEX11.dbd

Example B-5 lists the source of the secondary index DBD, SINDEX11, that is needed for both DBD AUTODB and PSB AUTPSB11. It is sourced and targeted on the segment ORDER.

Example B-5 SINDEX11 DBD source

```
DBD    NAME=SINDEX11,ACCESS=(INDEX,VSAM)
        DATASET  DD1=SINDX1P
        SEGM   NAME=SINDXA,PARENT=0,BYTES=66
        FIELD  NAME=(XFLDA,SEQ,U),BYTES=56,START=1,TYPE=C
        FIELD  NAME=DATE,BYTES=10,START=57,TYPE=C
        LCHILD NAME=(ORDER,AUTODB),INDEX=XFLD1
        DBDGEN
        FINISH
        END
```

B.2.5 SINDEX22.dbd

Example B-6 lists the source of the secondary index DBD, SINDEX22, that is needed for both DBD AUTODB and PSB AUTPSB11. It is sourced on segment MODEL and targeted on the root segment DEALER.

Example B-6 SINDEX22 DBD source

```
DBD    NAME=SINDEX22,ACCESS=(INDEX,VSAM)
        DATASET  DD1=SINDX2P
        SEGM   NAME=SINDXB,PARENT=0,BYTES=34
        FIELD  NAME=(XFLDB,SEQ,U),BYTES=28,START=1,TYPE=C
        FIELD  NAME=COUNT,BYTES=2,START=25,TYPE=C
        FIELD  NAME=ENQUIRS,BYTES=4,START=25,TYPE=P
        LCHILD NAME=(DEALER,AUTODB),INDEX=XFLD2
        DBDGEN
        FINISH
        END
```

B.2.6 AUTOLDB.dbd

Example B-7 lists the source of the logical DBD, AUTLDB, that is needed for PSB AUTPSB11. It is based on the physical DBD AUTODB and uses its two internal logical relationships and its logical relationship with the physical DBD EMPDB2.

Example B-7 AUTOLDB DBD source

```
DBD     NAME=AUTOLDB,ACCESS=LOGICAL
 DATASET  LOGICAL
 SEGM NAME=DEALER,PARENT=0,SOURCE=((DEALER,,AUTODB))
 SEGM NAME=MODEL,PARENT=DEALER,SOURCE=((MODEL,,AUTODB))
 SEGM NAME=ORDER,PARENT=MODEL,SOURCE=((ORDER,,AUTODB))
 SEGM NAME=SALES,PARENT=MODEL,                                          X
               SOURCE=((SALES,DATA,AUTODB),(STOCK,DATA,AUTODB))
 SEGM NAME=STOCK,PARENT=MODEL,SOURCE=((STOCK,,AUTODB))
 SEGM NAME=STOCSALE,PARENT=STOCK,                                       X
               SOURCE=((STOCSALE,DATA,AUTODB),(MODEL,KEY,AUTODB))
 SEGM NAME=SALESPER,PARENT=DEALER,                                      X
               SOURCE=((SALESPER,DATA,AUTODB),(EMPL,DATA,EMPDB2))
 SEGM NAME=SALESINF,PARENT=SALESPER,                                    X
               SOURCE=((SALESINF,,AUTODB))
 SEGM NAME=EMPLINFO,PARENT=SALESPER,                                    X
               SOURCE=((EMPLINFO,,EMPDB2))
```

```
DBDGEN
FINISH
END
```

B.2.7 EMPLDB2.dbd

Example B-8 lists the source of the logical DBD, EMPLDB, that is needed for PSB AUTPSB11. It is based on the physical DBD EMPDB2 and uses its logical relationship with the physical DBD AUTODB.

Example B-8 EMPLDB DBD source

```
DBD     NAME=EMPLDB2,ACCESS=LOGICAL
        DATASET LOGICAL
        SEGM    NAME=EMPL,PARENT=0,SOURCE=((EMPL,,EMPDB2))
        SEGM    NAME=DEALER,PARENT=EMPL,                                X
                SOURCE=((EMPSAL,KEY,EMPDB2),(DEALER,DATA,AUTODB))
        SEGM    NAME=SALESINF,PARENT=DEALER,                            X
                SOURCE=((SALESINF,,AUTODB))
        SEGM    NAME=EMPLINFO,PARENT=EMPL,                              X
                SOURCE=((EMPLINFO,,EMPDB2))
        DBDGEN
        FINISH
        END
```

C

The environment for our scenarios

This appendix gives information about the configuration that we used and the versions of the programs we used for the Open Database scenarios that are available as sources, as described in Appendix D, “Additional material” on page 251.

The topics that we discuss in this appendix are:

- Used system configuration
- Used application versions
- Suggested APAR numbers

C.1 Used system configuration

Our IMS system is an IMS Version 11 and is located on an LPAR called WTSC63. The DB2 for System z, which is used in one of the scenarios, is also located on the LPAR WTSC63.

The IMS was enhanced with an ODBM and an IMS Connect address spaces, as shown in Figure C-1. We omitted all of the other address spaces in the picture for clarity. Notice that the SCI connection from IMS Connect to ODBM is by program call (PC), if they are within the same LPAR, or through XCF if they are on a different LPAR. This is the normal behavior of SCI. If XA is used in a managed environment the application server will keep track of the transactionallity. On the z/OS side, this would involve RRS, which was therefore enabled in all used components.

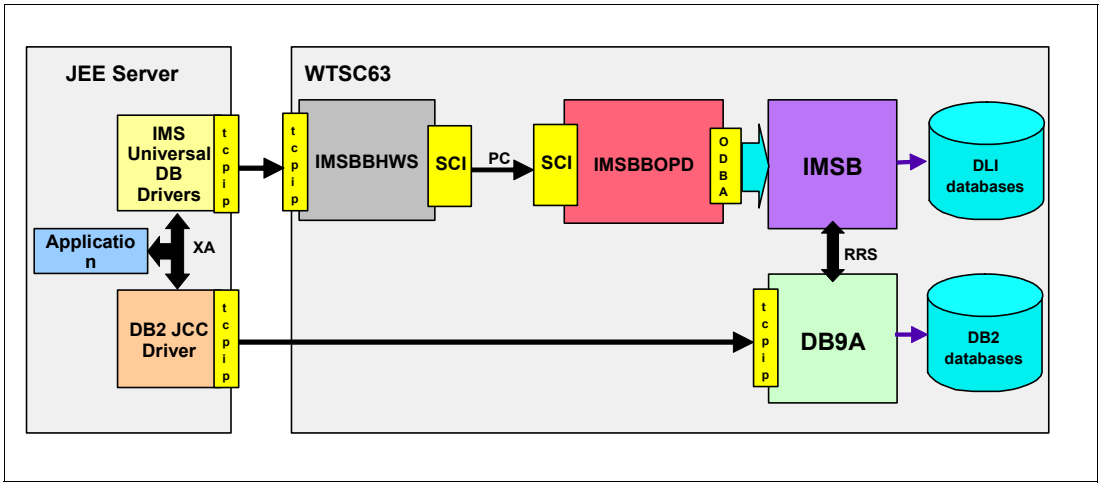


Figure C-1 The system configuration used for this book

For more informations about the used configuration see Chapter 3, “System environment” on page 43.

C.2 Used application versions

Table C-1 lists the versions of the products used in this book.

Table C-1 Products and versions

Product	Version
IBM Cognos Virtual View Manager	V 8.4.1
IBM Data Studio	V 2.2.0.1 (2.2.0.20091124_1634)
IBM Rational Application Developer for WebSphere Software	V 7.5.5 (7.5.5.20091203_0703)
IBM Rational Developer for System z	V 7.6.0.1 (7.6.0.20091216_1622)
WebSphere Application Server Version 7.0	V 7.0.0.7 (2.0.1.20091203_0240)
IMS Enterprise Suite DLIModel Utility Plug-in	V 2.0.3.20091102_1145 with DLIModel Utility Fix V (2.0.3.20100209_1546)
IBM Installation Manager	V 1.3.4

Product	Version
IBM Mashup Center	V2.0
IMS Universal Drivers	see required APARs
Java Development Kit (JDK)	IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Windows XP x86-32 jvmwi3260sr6-20091001_43491
DB2 JCC Driver	V 3.57 / V 4.7

C.3 Suggested APAR numbers

Table C-2 lists the suggested APARs, based on what we applied in our environment in order to have working samples, and what has recently being planned in the Open Database area.

Table C-2 Products and APARs

Product/function	APARs	PTF
SQL exception code cleanup/doc		
Open Database universal drivers now support type-2 connectivity to IMS DB within IMS JMP and JBP regions	PK86498	UK57311 UK57312
Universal driver enhancement for COBOL PIC G datatype	PK98486	UK54419
Open Database universal driver support extended to type-2 or local access of IMS database from WebSphere Application Server for z/OS	PK99686	OPEN
Open Database universal driver support extended to type-2 or local access of IMS database. This new driver allows the universal drivers to be used from WebSphere Application Server for z/OS, DB2 z/OS stored procedures, CICS and IMS java dependent regions (JDR). As part of the JDR, a new API has also been implemented	PM02734	UK57317
IMS Connect Extensions. Improved use of OMEGAMON to capture the event buffers	PM04643	UK55059
After an IMS /ERE, ODBM fails to reconnect to IMS (use the UPD ODBM START(CONNECTION) DATASTORE() command)	PM06073	OPEN

Product/function	APARs	PTF
IMS Problem Investigator. New function to track and format CEX event records for ODBA transactions.	PM09535	UK56453
IMS 11 generic JDBC support (for Cognos and Data Source Explorer integration)	PM12893	OPEN
Open Database universal driver support extended to type-2 or local access of IMS database from WebSphere Application Server for z/OS	PM13216	OPEN
ODBM hangs during shutdown (Must use z/OS Cancel command to shutdown ODBM	PM13448	OPEN

D

Additional material

In this chapter, we discuss additional material that you can download from the Internet.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG247856`

Alternatively, you can go to the IBM Redbooks web site at:

`ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247856.

Using the web material

The additional web material that accompanies this book includes the following files:

File name	***Description***
AUTPSB11.jar	It contains the necessary AUTPSB11DatabaseView Java class in the package samples.ims.openDb generated by the IMS Enterprise Suite DLIModel Utility, as well as all other generated artefacts like the XML schemas.
AUTPSB11Modified.jar	It contains the same as the AUTPSB11.jar, but the DatabaseView is modified to match the PSB and DBD special data types. To use this class you have to delete the contents of the IVP CarDealer Database and insert data with the corresponding representation (Packeddecimal and Hexadecimal).

CarDealer_DBDandPSBSources.zip
It is a zip file and contains the DBDs and the PSB of the Car Dealer IVP Database which are used for this example.

IMSandDB2.ear It is the Enterprise Archive file for the deployment to a WebSphere Application Server Version 7. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudbJXA.rar contents, JDK, WebSphere Application Server 7 runtime and AUTPSB11.jar). For installing it in WebSphere Application Server you have to install the IMS Universal JDBC XA resource adapter and the IBM DB2 Data Server Driver as described in the scenario.

IMSOpenDBTransaction.zip
It is the project interchange file for the IMS Transaction example. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudb.jar, imsutm.jar, JDK and AUTPSB11.jar).

IMSOpenDBApp.zip It is the project interchange file for the IMS Standalone examples and more self explaining stand-alone examples. When you import this file to your Eclipse, you must add the required libraries to your Java Build path (imsudb.jar, JDK and AUTPSB11.jar).

IMSOpenDBSource.zip
It is the project interchange file for the IMS Managed environment examples and the used code fragments in the book. When you import this file to your Eclipse, you must add the required libraries to your Java Build path (imsudbJXA.rar contents, JDK, WebSphere Application Server 7 runtime and AUTPSB11.jar).

dlitest1 It is an example of using the IMS Universal DL/I driver to retrieve, update, and delete rows in the IMS sample DB AUTODB database and the PCB that references it in the PSB AUTPSB11.

dlitest2 It is an example of using the batch method feature of the IMS Universal DL/I driver to retrieve, update, and delete rows in the IMS sample DB AUTODB database and the PCB that references it in the PSB AUTPSB11. Over 8000 segments were added to the sample data supplied with the IMS product for this test.

System requirements for downloading the web material

The following system configuration is recommended:

Hard disk space: 2 MB minimum
Operating System: Windows XP or alternatively Linux
Processor: x86 architecture
Memory: 256 MB minimum

How to use the web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the web material zip file into this folder if you want to view the code.

Tip: Jar files are also packed archives and can be extracted by renaming the file extension to .zip.

The Project Interchange files can be imported in Eclipse by using the **File** → **Import** and select Project Interchange file. The necessary build path must be configured so that the reference errors disappear; therefore, you need, additionally, the IMS Universal DB Drivers and other resources, such as the WebSphere Application Server Runtime and Java Development Kit.

Abbreviations and acronyms

ACEE	access control environment element
AGN	Application Group Name
AIB	Application Interface Block
AOI	Automated Operator Interface
APF	authorized program facility
API	application programming interface
APPC	Advanced Program-to-Program Communication
APSB	Allocate PSB
ARM	Automatic Restart Manager
ASN	Abend Search and Notification
BMP	batch messaging program
BPE	Base Primitive Environment
CBM	Component Business Modeling
CCF	Common Connector Framework
CCI	Common Client Interface
CDE	contents directory entry
CGI	Common Gateway Interface
CI	Control Interval
CICS	Customer Information Control System
CLI	command line interface
CLP	command line processor
CLS	Common Service Layer
CSL	Common Service Layer
CSM	Complete Status Message
CTG	CICS Transaction Gateway
DB2	Database 2
DBCTL	Database Control
DBD	database description
DBMS	database management system
DBPCB	database PCB
DBRA	Database Resource Adapter
DBRC	Database Recovery Control
DDM	Distributed Data Management
DEDB	data entry database
DLET	Delete
DMB	data management block
DRA	Database Resource Adapter
DRDA	Distributed Relational Database Architecture
DSN	Data Source Name
DTP	distributed transaction processing
EAB	Enterprise Access Builder
EAI	enterprise application integration
ECB	Event Control Block
EIS	enterprise information systems
EISS	Enterprise Information Systems
EISs	Enterprise Information Systems
EJB	Enterprise JavaBean
EJBs	Enterprise JavaBeans
EMF	Eclipse Modeling Framework
ETO	Extended Terminal Option
EWLM	Enterprise Workload Manager™
GEF	Graphical Editor Framework
GN	Get Next
GNP	Get Next within Parent
GU	Get Unique
GUI	graphical user interface
HALDB	High Availability Large Data Bases
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines Corporation
IDE	Integrated Development Environment
IDS	Indormix Dynamic Server
ILDS	indirect list data set
IMS	Information Management System
IPCS	Interactive Problem Control System
IPL	initial program load
IRLM	internal resource lock manager
IRM	IMS request message
ISC	intersystem communication
ISPF	Interactive Systems Productivity Facility
ISRT	Insert
ISVs	independent software vendors
ITSO	International Technical Support Organization

IVP	Installation Verification Procedure
IVPEX	IVP Export Utility
J2C	Java EE Connector Architecture
JEE	Java Platform, Enterprise Edition
JBP	Java Batch Program
JCA	Java Connector Architecture
JCC	JAVA Common Client
JCL	job control language
JDBC	Java Database Connectivity
JDK	Java Development Kit
JMP	Java Message Program
JMS	Java Message Server
JNDI	Java Naming and Directory Interface
JRE	Java Runtime Environment
JSP	JavaServer Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
Java EE	Java Platform Enterprise Edition
KBLA	Knowledge-Based Log Analysis
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LPAR	logical partition
LSQA	local system queue area
LTERM	logical terminal
LU	logical unit
LU2	logical unit 2
MCI	Message Control Information
MDB	message-driven bean
MFS	Message Format Service
MOD	message output descriptor
MPP	message processing program
MSC	Multiple Systems Coupling
MVS	Multiple Virtual System
ODBA	Open Database Access
ODBC	Open DataBase Connectivity
ODBM	Open Database Manager
OLDS	online log data set
OLDSs	online log data sets
OLR	Online Reorganization
OM	Operations Manager
OO	object-oriented
OTMA C/I	OTMA Callable Interface
OTMA	Open Transaction Manager Access
PC	program call
PCB	program communication block
PCBs	Program Control Blocks
PDU	Partition Definition Utility
PPT	program properties table
PRA	Parallel RECON Access
PSB	program specification block
PSBs	program specification blocks
RACF	Resource Access Control Facility
RAD	Rational Application Developer
RAR	resource adapter archive
RAS	Resource Access Security
RDDS	Resource Definition Data Set
RECON	Recovery Control
REPL	Replace
RM	resource manager
RMM	Request MOD Message
RRS	Resource Recovery Services
RRS/MVS	Resource Recovery Services/MVS
RSM	request status message
RYO	Roll-Your-Own
SAF	System Authorization Facility
SCI	Structured Call Interface
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SLSB	stateless session bean
SMP/E	System Modification Program/Extended
SMU	Security Maintenance Utility
SNA	Systems Network Architecture
SOA	service-oriented architecture
SOMA	Service Oriented Modeling and Architecture
SPE	small programming enhancement
SPOC	single point of control
SQLJ	SQL for Java
SSAs	segment search arguments
SSL	Secure Socket Layer
SSPM	Sysplex serialized program management
STE	storage tracking element
STSN	set and test sequence numbers

SVL	Silicon Valley Laboratories
TCO	Time-Controlled Operations
TCP/IP	Transmission Control Protocol/Internet Protocol
THREAD	TYPE
TMRA	Transaction Manager Resource Adapter
TPIPE	transaction pipe
TRACE	TYPE
UOR	Unit of Recovery
VIPA	Virtual IP Addressing
VVM	Virtual View Manager
W3C	World Wide Web Consortium
WAN	wide area network
WID	WebSphere Integration Developer
WLM	workload manager
WPS	WebSphere Process Server
WSDL	Web Service Description Language
WWW	World Wide Web
XCF	cross-system coupling facility
XMI	XML Metadata Interchange
XML	Extensible Markup Language
db2cli	DB2 Interactive Call Level Interface

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 260. Note that some of the documents referenced here may be available in softcopy only.

- *IMS Version 11 Technical Overview,* SG24-7807
- *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01
- *Powering SOA Solutions with IMS*, SG24-7662
- *Powering SOA with IBM Data Servers,* SG24-7259
- *IMS Performance and Tuning Guide,* SG24-7324
- *IMS V10 Implementation Guide: A Technical Overview,* SG24-7526
- *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity,* SG24-6794
- *Publishing IMS and DB2 Data Using WebSphere Information Integrator: Configuration and Monitoring Guide,* SG24-7132
- *IMS V9 Implementation Guide: A Technical Overview,* SG24-6398
- *IMS V8 Implementation Guide: A Technical Introduction of the New Features,* SG24-6594
- *Reorganizing Databases Using IMS Tools: A Detailed Look at the IBM IMS High Performance Tools,* SG24-6074
- *IMS Installation and Maintenance Processes,* SG24-6574
- *Using IMS Data Management Tools for Fast Path Databases,* SG24-6866
- *IMS in the Parallel Sysplex Volume I: Reviewing the IMSplex Technology,* SG24-6908
- *IMS in the Parallel Sysplex Volume II: Planning the IMSplex,* SG24-6928

Other publications

These publications are also relevant as further information sources:

- *IMS Version 11 Application Programming*, SC19-2428
- *IMS Version 11 Application Programming APIs*, SC19-2429
- *IMS Version 11 Commands, Volume 1: IMS Commands*, SC19-2430
- *IMS Version 11 Commands, Volume 2: IMS Commands N-V*, SC19-2431
- *IMS Version 11 Commands, Volume 3: IMS Component and z/OS Commands,* SC19-2432
- *IMS Version 11 Communications and Connections*, SC19-2433
- *IMS Version 11 Database Administration*, SC19-2434

- *IMS Version 11 Database Utilities*, SC19-2435
- *IMS Version 11 Diagnosis*, GC19-2436
- *IMS Version 11 Exit Routines*, SC19-2437
- *IMS Version 11 Installation*, GC19-2438
- *IMS Version 11 Master Index and Glossary*, SC19-2440
- *IMS Messages and Codes, Volume 1: DFS Messages*, GC18-9712
- *IMS Messages and Codes, Volume 2: Non-DFS Messages*, GC18-9713
- *IMS Messages and Codes, Volume 3: IMS Abend Codes*, GC18-9714
- *IMS Messages and Codes, Volume 4: IMS Component Codes*, GC18-9715
- *IMS Version 11 Operations and Automation*, SC19-2441
- *IMS Version 11 Release Planning*, GC19-2442
- *IMS Version 11 System Administration*, SC19-2443
- *IMS Version 11 System Definition*, GC19-2444
- *IMS Version 11 System Programming APIs* , SC19-2445
- *IMS Version 11 System Utilities*, SC19-2446
- *IMS and SOA Executive Overview*, GC19-2516
- *IMS TM Resource Adapter User's Guide and Reference*, SC19-1211
- *Program Directory for Information Management System Transaction and Database Servers V11.0*, GI10-8788
- *IRLM Messages and Codes,* GC19-2666

Online resources

These web sites are also relevant as further information sources:

- IMS products and tools

  http://www.ibm.com/ims
- IMS Information Center

  http://publib.boulder.ibm.com/infocenter/imzic
- The IMS Enterprise Suite

  http://www.ibm.com/software/data/ims/soa-integration-suite/
- System z hardware

  http://www.ibm.com/systems/z/hardware/
- IMS tools:

  http://www.ibm.com/software/data/db2imstools/products/ims-tools.html

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

J

L

M

O

P

Q

R

S

T

U

V

W

X

Z

IMS 11 Open Database

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

IMS 11 Open Database

Install IMS Open Database and its prerequisites

Implement Java client access to IMS and DB2 data

Integrate Mash up Center with IMS Open Database

IMS Version 11 continues to provide the leadership in performance, reliability, and security that is expected from the product of choice for critical online operational applications. IMS 11 also offers new functions to help you keep pace with the evolving IT industry.

Through the introduction of the new IMS Enterprise Suite application developers with minimal knowledge of IMS Connect can start developing client applications to communicate with IMS.

With Open Database, IMS 11 also provides direct SQL access to IMS data from programs that run on any distributed platform, unlocking DL/I data to the world of SQL application programmers.

In this IBM Redbooks publication, system programmers get the steps for installing the new IMS components, and the application programmer can follow scenarios of how client applications can take advantage of SQL to access IMS data.

We describe the installation of prerequisites, such as IMS Connect and the Structured Call Interface component of Common Service Layer address space and document the set up of the three new IMS drivers:

- Universal DB resource adapter
- Universal JDBC driver
- Universal DL/I driver

Our scenarios use the JDBC driver for type-4 access from Windows® to a remote DL/I database and DB2 tables and extend it to use IBM Mashup Center to provide an effective web interface and to integrate with Open Database.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7856-00 ISBN 0738434566